Universidad Carlos III de Madrid

Escuela politécnica superior

Departamento de Telemática

Ingeniería técnica de Telecomunicación especialidad Sonido e Imagen

Proyecto de Fin de Carrera

Integración de un sistema de televigilancia a través de de un dispositivo móvil en un servidor Web

Autor: Juan Galán García

Tutor: Mario Muñoz Organero

Noviembre 2009

Título: Integración de un sistema de televigilancia a través de un dispositivo móvil en un servidor Web.

Autor: Juan Galán García

Tutor: Mario Muñoz Organero

EL TRIBUNAL

Presidente:

Secretario:

Vocal:

Realizado el acto de defensa del Proyecto Fin de Carrera el día __ de _____ de 2009 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de:

Fdo: Presidente Fdo: Secretario Fdo: Vocal

Resumen

El Proyecto Fin de Carrera "*Integración de un sistema de televigilancia a través de un dispositivo móvil en un servidor Web*" analiza e implementa un servidor Web de apoyo a un sistema de televigilancia instalado en un dispositivo móvil.

Las funciones principales que implementa este sistema son: conexión del dispositivo móvil al servidor Web, almacenamiento de imágenes enviadas desde el teléfono móvil en un servidor Web y la visualización de las mismas en un navegador Web. Como última funcionalidad el servidor Web generará un correo electrónico con la imagen recibida en el servidor.

El principal objetivo de este proyecto es dotar de valor añadido a un sistema de televigilancia diseñado para funcionar en teléfonos móviles de tal forma que el usuario pueda acceder a la información enviada por su teléfono móvil desde cualquier punto del planeta con el único requisito de tener una conexión a Internet.

Agradecimientos

Este proyecto no habría sido posible sin la inestimable colaboración de mis padres. Gracias.

No puedo obviar en este epígrafe a mi tutor Mario Muñoz, por haber aportado la idea, sugerencias, opiniones, comentarios y sobre todo por su paciencia.

Agradezco igualmente su apoyo, aguante y dedicación a Sonia y a todos los compañeros de la carrera: Camarma, Clara, Joven Herrera, Alvarito, Sarita, Lucia, Cris, Carlos, Alex, Toni, Rubén, Dani, Golfo, Celia, Elma, Patri y todos los que se me olvidan. Agradezco también a todos los que lo intentaron pero no llegaron en especial al Soriano por sus buenos momentos allá en primero de carrera.

Por supuesto agradecer y brindar este proyecto a mis amigos por aguantarme y intentar entenderme: Chechu, Yago, Héctor, Lucas, Iñigo, Miki, Mario, Sergio, Jose, Luis, Mauro, Jon, Jonatan y seguro que se me olvida alguno más pero vosotros sabéis quienes sois.

Por ultimo dedicar el proyecto a todos los que de alguna manera lo han apoyado o ayudado a conseguirlo, profesores, compañeros, camareros, todos sois en parte responsables del éxito o el fracaso de esta difícil empresa.

Muchas gracias a todos

Índice de contenidos

Índice de imágenes

1. Introducción y objetivos.

1.1. Introducción.

Este proyecto se encuentra enmarcado dentro del proyecto global "*Desarrollo e integración de un sistema de televigilancia a través de un dispositivo móvil en un servidor Web*".

El propósito general del proyecto global es la implementación de un sistema de televigilancia que pueda ser instalado en un dispositivo móvil, especialmente teléfonos móviles, y que la información que se genere pueda ser visualizada por el usuario en tiempo real a través del envío de mensajes multimedia u otras opciones.

La motivación principal de realizar el proyecto "Integración de un sistema de televigilancia a través de un dispositivo móvil en un servidor Web" será la de analizar las diferentes opciones que se presentan para visualizar la información enviada por un teléfono móvil, así como dotar de un valor añadido a una aplicación de televigilancia instalada en un dispositivo móvil.

En este proyecto se comentan y analizan los siguientes aspectos:

- Posibles casos de uso de la aplicación.(*Capitulo 2*)
- Elementos de los que consta el sistema implementado y descripción a alto nivel de los mismos. (*Capitulo 3*)
- Descripción de las tecnologías en liza para una correcta implementación del sistema. (*Capitulo 4*)
- Análisis funcional y a bajo nivel del sistema implementado. (*Capitulo 5*)
- Breve reseña a los plazos de ejecución del proyecto así como su coste. (*Capitulo 7*)
- Posibles avances y mejoras de la aplicación. (*Capitulo 8*)

El estudio, desarrollo y explicación de la aplicación móvil se encuentra en el texto **"***Desarrollo de un sistema de televigilancia a través de un dispositivo móvil en un servidor Web***"**, llevado a cabo por Javier Rodríguez Camarma.

1.2. Objetivos.

El principal objetivo de este proyecto es implementar un servidor Web que sirva de apoyo a un sistema de videovigilancia instalada en un dispositivo de la que ha sido objeto otro proyecto en este mismo departamento. Dicho servidor debe recibir los datos enviados por el dispositivo móvil y mostrarlos siempre que el usuario desee acceder a ellos. El acceso al servidor Web y los datos almacenados en él debe realizarse de manera segura y personalizada, para ello se debe implementar un sistema de autenticación mediante el cual el usuario podrá acceder a los datos enviados por su dispositivo móvil.

2. Posibles casos de uso del sistema.

El sistema a implementar se ha diseñado principalmente para cubrir un único caso de uso:

1. El usuario inicia la aplicación de video vigilancia en el dispositivo móvil antes de salir del lugar donde desee realizar la labor de vigilancia. Una vez conectado el sistema iniciará su labor enviando datos al servidor Web, en este caso imágenes, que el usuario podrá ver desde cualquier dispositivo con conexión a Internet a través de una página Web o el correo electrónico.

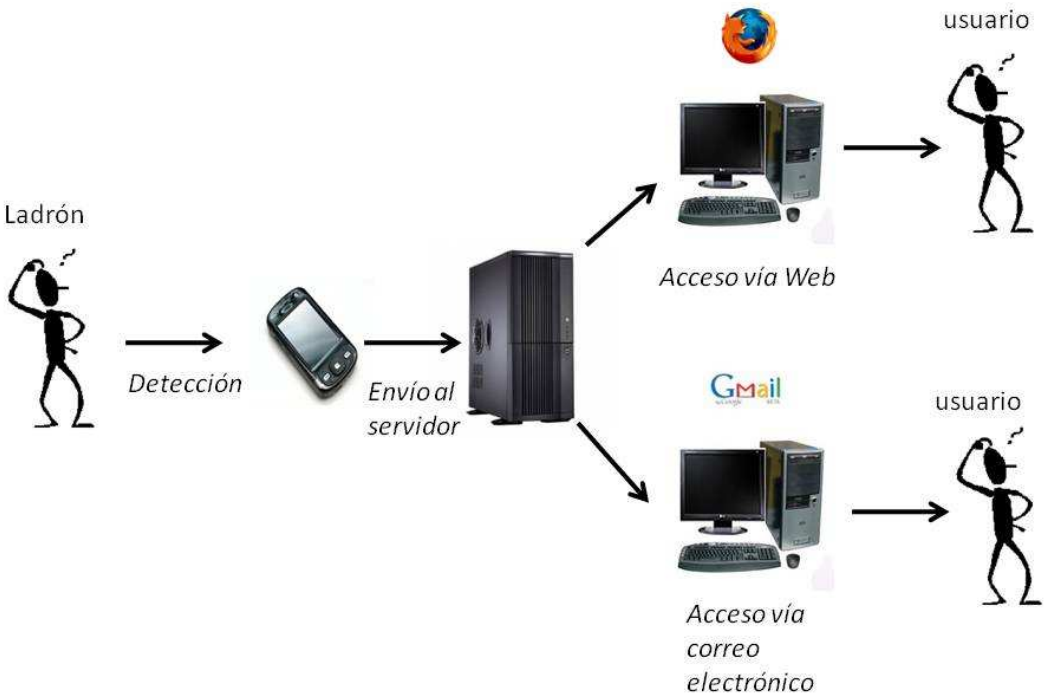


Figura 1: Diagrama de casos de uso de la aplicación

3. Introducción al sistema implementado.

El sistema desarrollado en este proyecto se compone fundamentalmente de los siguientes elementos:

- Aplicación instalada en un teléfono móvil que detecta movimiento, capturando la imagen y enviando esta a través de Internet a un servidor Web. La mayor parte de esta aplicación ha sido desarrollada de forma paralela en otro proyecto, siendo únicamente objeto de este sistema lo relativo a la conexión y envío de imágenes a través de Internet.
- Aplicación Web mediante la cual se podrá acceder a las imágenes mandadas por el teléfono móvil.
- Servidor Web que se encargará de recibir la imagen enviada por el teléfono móvil y almacenarla o mostrarla cuando el usuario la solicite. También se encargará de funciones de autenticación tanto del usuario del teléfono móvil como del usuario Web, así como el envío de un mensaje de correo electrónico con la imagen procedente del teléfono móvil.

El sistema desarrollado sigue un modelo de desarrollo cliente-servidor en la cual actuarán como cliente el teléfono móvil y el navegador Web. Como servidor se usará un servidor Web Apache-Tomcat. El conjunto del sistema utiliza un patrón MVC (*Modelo-Vista-Controlador*).

3.1. Cliente Móvil.

La aplicación instalada en el teléfono móvil, en primer lugar, comprobará si el usuario desea activar el uso de Internet para el envío de imágenes, en cuyo caso, la imagen resultante de una detección de movimiento se enviará al servidor Web mediante una petición HTTP. En caso de no aceptarse el uso de Internet la aplicación funcionará tal y como se describe en el proyecto que desarrolla la aplicación móvil.

Para poder esta funcionalidad el usuario deberá estar registrado previamente en el sistema. Se deberá introducir el número de teléfono móvil con el cual nos hemos

registrado, éste se enviará al servidor que comprobará si existe en la base de datos un usuario registrado con ese número de teléfono, en cuyo caso podremos acceder a la aplicación. En caso contrario se mostrará al usuario un mensaje de error.

La aplicación móvil está desarrollada en lenguaje Java, más concretamente J2ME (*Java 2 Micro Edition*). La conexión a Internet y el envío de imágenes se consigue gracias al paquete *javax.microedition.io* incluido en la especificación de J2ME. Todas las peticiones al servidor se realizarán mediante el protocolo HTTP. El servidor utilizado será Apache-Tomcat.

En el momento en el cual el servidor Web recibe una imagen, la almacena en un fichero y inserta en la base de datos información relativa a la imagen capturada tal como el número de teléfono que la ha generado, la fecha y hora en la cual se generó y la ruta donde la imagen se almacenará en el servidor.

Paralelamente al almacenamiento de la imagen, el servidor creará un mensaje de correo electrónico gracias a la API JavaMail, incluida en la especificación de Java utilizada. El mensaje se enviará mediante el protocolo SMTP (*Simple Mail Transfer Protocol*) a través del servidor de correo electrónico GMail (http://www.gmail.com).

El funcionamiento del sistema podría resumirse con el siguiente diagrama:

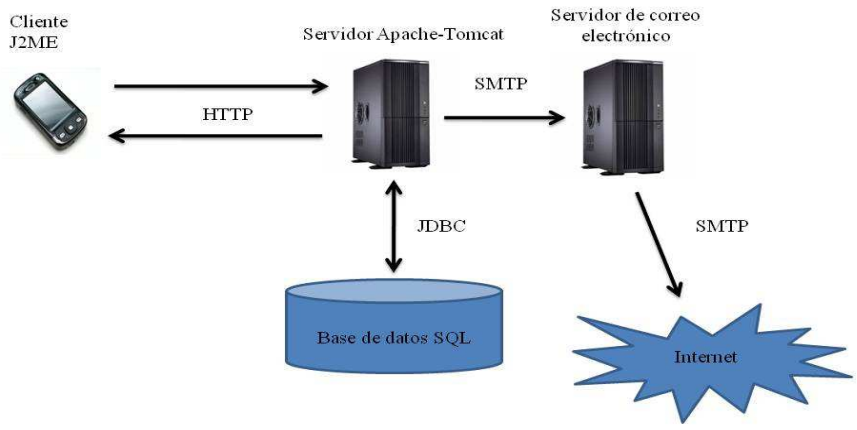


Figura 2: Funcionamiento del sistema

3.2. Cliente Web.

En este caso, se considera como cliente Web un navegador. El sistema se ha optimizado y probado en el navegador Web Mozilla Firefox (*http://www.mozilla-europe.org/es/firefox*).

El usuario deberá autenticarse en el servidor, para ello deberá introducir un nombre de usuario y una contraseña. Si es la primera vez que se accede al servicio el usuario deberá registrarse en la base de datos introduciendo como datos para el registro su nombre de usuario, contraseña, número de teléfono móvil (Para el registro del usuario en la aplicación móvil) y dirección de correo electrónico (A la cual se mandarán las imágenes enviadas por el teléfono móvil).

Una vez registrado y autenticado, podremos acceder al servicio y a las imágenes que ha mandado la aplicación instalada en el teléfono móvil. La aplicación Web interactúa con el servidor mediante peticiones AJAX.

3.3. Servidor Web

El servidor Web se encargará de recibir toda la información procedente tanto de la aplicación instalada en el teléfono móvil como del navegador Web, registrarla en la base de datos y servir la página deseada al usuario.

El servidor Web utilizado será un servidor Apache-Tomcat. Para la implementación del servidor se ha utilizado tecnología Java, más concretamente Servlets y JSP. La base de datos será MySQL.

La interacción entre cliente y servidor se podrá aproximar mediante el siguiente esquema.

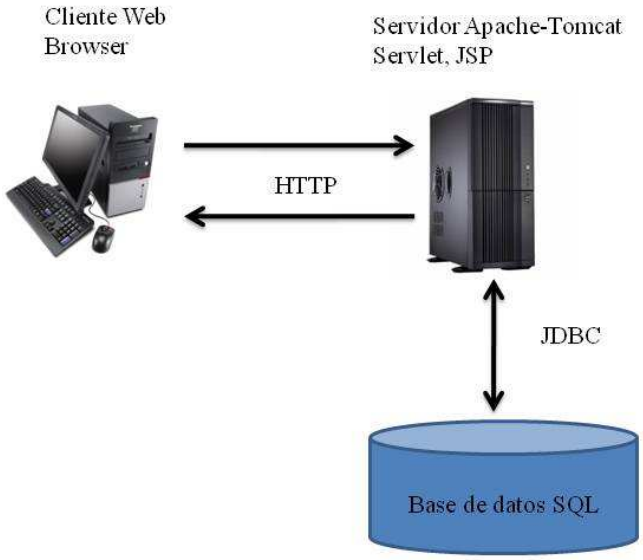


Figura 3: Interacción cliente-servidor

4. Descripción de las tecnologías utilizadas.

4.1. Arquitectura cliente-servidor

Una arquitectura cliente-servidor consiste básicamente en un cliente que realiza peticiones a otro programa (servidor) que le da respuesta. En esta arquitectura la capacidad de proceso está repartida entre los clientes y los servidores, aunque son más importantes las ventajas de tipo organizativo debido a la centralización de la gestión de la información y la separación de responsabilidades, lo que facilita y clarifica el diseño del sistema.

La separación entre cliente y servidor es una separación de tipo lógico, donde el servidor no se ejecuta únicamente en una sola maquina ni es un solo programa. Los tipos específicos de servidor son por ejemplo servidores Web, servidores de archivos o servidores de correo electrónico. Mientras que sus propósitos varían de un servidor a otro, la arquitectura básica seguirá siendo la misma.

En la arquitectura cliente-servidor el **cliente** es el remitente de la solicitud. Sus características principales son:

- Es quien inicia las peticiones, tiene un papel activo en la comunicación.
- Espera y recibe respuestas del servidor.
- Puede conectarse a varios servidores a la vez.

El receptor de la solicitud enviada por el cliente se conoce como **servidor.** Sus características son las siguientes:

- Al iniciarse esperan a que lleguen peticiones desde el cliente, por tanto desempeñan un papel pasivo en la comunicación.
- Tras la recepción de una petición, la procesan y luego envían la respuesta al cliente.
- Aceptan conexiones desde un gran número de clientes.

4.2. Patrón MVC

Cuando en una aplicación coexiste código de control, acceso a datos y presentación, pueden producirse multitud de problemas tales como:

— Fuertes repercusiones en la aplicación al introducir cambios en el código.

— No posibilidad de reutilizar código debido al alto acoplamiento de clases.

— Añadir nuevos datos puede implicar un cambio en la lógica de la aplicación.

El patrón MVC (*Modelo-Vista-Controlador*) mediante la disociación del acceso a datos, la lógica de control y la presentación de los datos. Las diferentes partes de las que consta este patrón serían:

— Modelo: Datos que se manipulan y se muestran (Datos contenidos en una base de datos).

— Vista: Lo que se visualiza en la pantalla (Páginas HTML).

— Controlador: Gestiona la petición y decide que lógica invocar y que vista mostrar (Servlet).

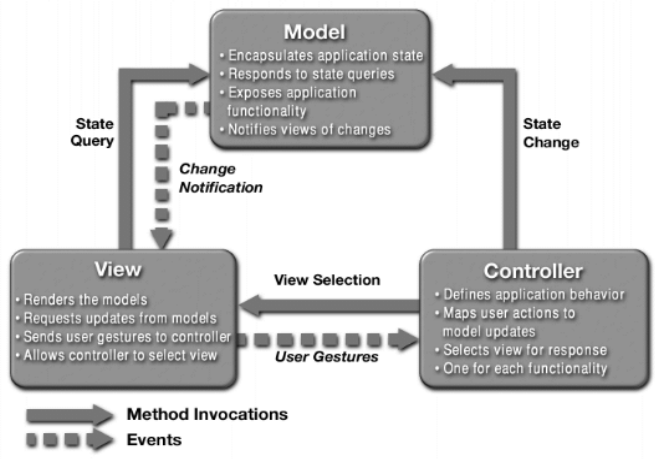


Figura 4: Esquema del patrón MVC

Las consecuencias de usar este patrón de diseño son:

— Reutilización de los componentes del modelo.

— Aumento de la complejidad del diseño.

— Apoyo más sencillo a nuevos tipos de clientes.

4.3. Protocolo HTTP

El protocolo http (*Hyper Text Transfer Protocol*) es un protocolo del nivel de aplicación para sistemas de información multimedia distribuidos. Entre las propiedades de este protocolo se pueden destacar las siguientes:

— Esquema de direccionamiento comprensible: Se utiliza URI (Universal Resource Identifier) para localizar sitios (URL) o nombres (URN) sobre los que se aplicará una petición. La forma general de un URL es *protocolo://host:puerto/fichero.extensión.* Por ejemplo si se tiene la URL http://midominio:8080/mipagina.html se estaría pidiendo una página HTML a la maquina llamada *midominio* por el puerto 8080 mediante el protocolo http.

— Arquitectura cliente-servidor: Este protocolo se ajusta al paradigma petición/respuesta. La comunicación se establece sobre el protocolo TCP/IP. Por defecto se usa el puerto 80 aunque puede utilizarse otro.

— Protocolo no orientado a conexión y sin estado: Una vez que el servidor responde a una petición realizada desde el cliente se rompe la conexión entre ambos. Además no se guarda el contexto de la conexión para siguientes conexiones.

— Abierto a otros tipos de datos: http utiliza tipos MIME (*Multipart Internet Mail Extension*) para determinar el tipo de datos que transporta. Una vez que el servidor http envía la respuesta a una petición desde el cliente, incluye una cabecera que le indica el tipo de datos que la componen, será el cliente el encargado de gestionar esos datos.

Una petición HTTP está compuesta por una cabecera y opcionalmente los datos. En la cabecera se especifica el método de la petición, los tipos de datos devueltos o un código de estado. Los 2 métodos de la petición que se usarán en este proyecto son:

— Método GET: Se utiliza tanto para recuperar información identificada por un URI por parte de los navegadores como para enviar una pequeña cantidad de información al servidor en pares *atributo-valor* añadidos detrás de la URI detrás de un símbolo de interrogación (GET cgi/saludar.pl?nombre=juan&apellido=galan http/1.0). La longitud de la petición GET está limitada por el espacio libre en los buffers de entrada.

— Método POST: Se refiere normalmente a la invocación de procesos que generan datos que serán devueltos como respuesta a la petición. Se utiliza también para aportar datos de entrada a esos programas, aunque en este caso los pares *atributo-valor* van incluidos en el cuerpo de la petición. De este modo las peticiones POST no sufren limitación de espacio y puede enviar mucha más información al servidor.

El servidor http responde al cliente con un código que informa sobre el estado de la petición. Los códigos se agrupan según las siguientes categorías.

Rango	**Significado**
100-199	Informativo
200-299	Éxito en la petición
300-399	Petición redirigida, necesarias más acciones
400-499	Petición incompleta
500-599	Errores en el servidor

Figura 5: Códigos de estado HTTP

4.4. Tecnologías del lado del cliente

4.4.1. AJAX

AJAX (*Asynchronus Javascript and XML*) no es una tecnología en sí misma, en realidad se trata de varias tecnologías unidas para lograr un determinado resultado. Las tecnologías que usa AJAX son:

- XHTML y CSS para crear una presentación basada en estándares.
- DOM para la interacción y manipulación dinámica de la presentación.
- XML, XSL y JSON para el intercambio y manipulación de la información.
- XMLHttpRequest para el intercambio y la manipulación de información.
- Javascript para unir todas las tecnologías.

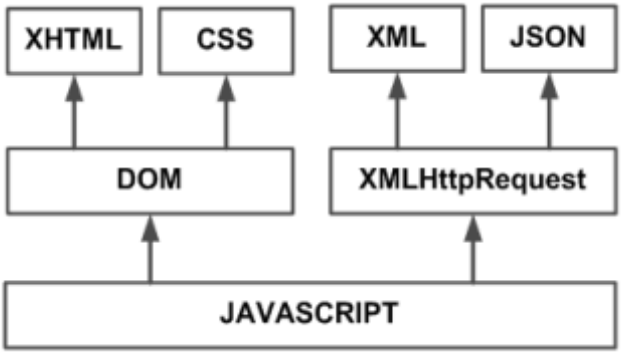


Figura 6: Esquema de tecnologías presentes en AJAX

AJAX permite mejorar la interacción del usuario con la aplicación, evitando recargas constantes de las paginas ya que la comunicación con el servidor se produce en un segundo plano. Esto se realiza mediante la creación de un elemento intermedio entre el cliente y el servidor. Esta nueva capa intermedia mejora la respuesta de la aplicación, ya que el usuario nunca se encuentra la ventana del navegador vacía a la espera de una respuesta del servidor. El siguiente esquema muestra la diferencia más importante entre una aplicación Web tradicional y una aplicación Web creada con AJAX. La imagen superior muestra la interacción síncrona propia de las aplicaciones Web tradicionales.

La imagen inferior muestra la comunicación asíncrona de las aplicaciones creadas con AJAX.

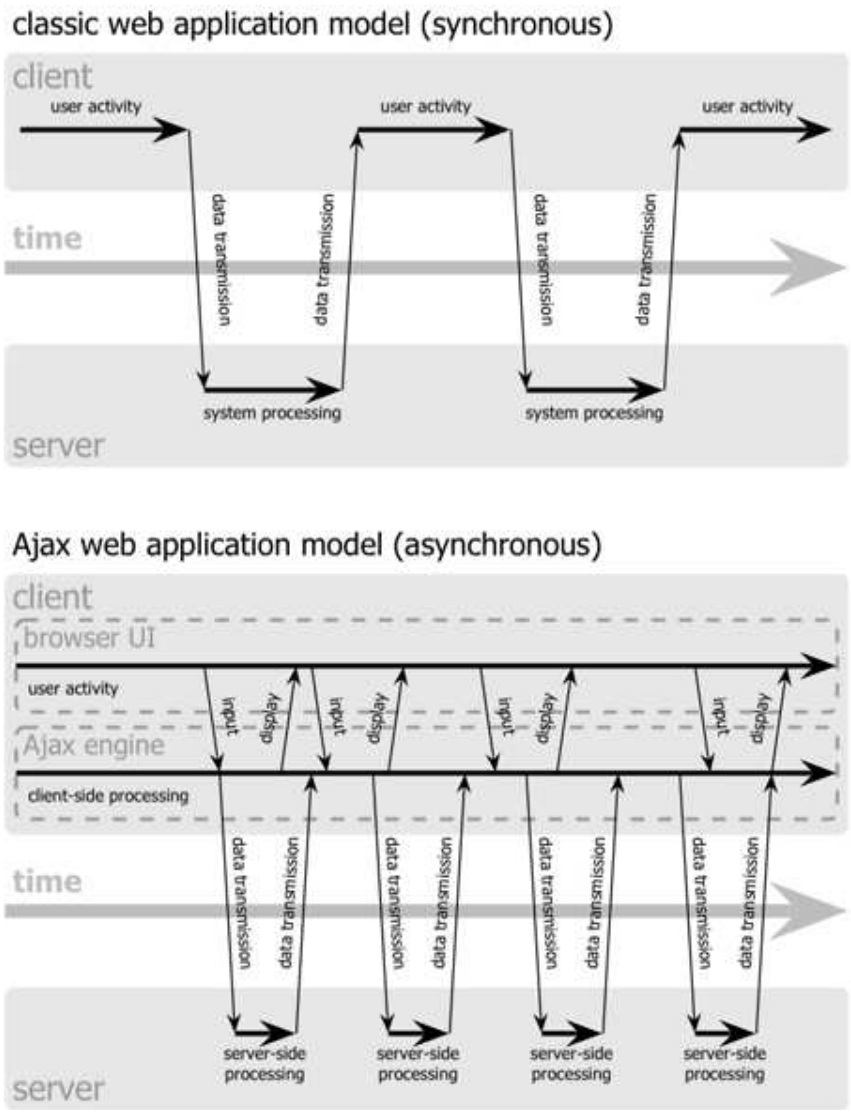


Figura 7: Diferencias entre la comunicación síncrona y asíncrona

Las peticiones HTTP al servidor se sustituyen por peticiones Javascript que se realizan al elemento encargado de AJAX. Las peticiones más simples no requieren intervención del servidor por lo que la respuesta es inmediata. Si la interacción requiere una respuesta del servidor, la petición se realiza de forma asíncrona mediante AJAX.

Se profundizará más sobre AJAX en el capítulo 5 de este proyecto.

4.4.2. XHTML

XHTML (*Extensible Hypertext Markup Language*) es una versión más limpia de HTML, que nace con el objetivo de reemplazar a HTML ante su limitación de uso con las cada vez más abundantes herramientas basadas en XML. XHTML extiende HTML 4.0 combinando la sintaxis de HTML, diseñado para mostrar datos, y XML, diseñado para describir los datos.

XHTML surge como el lenguaje cuyo etiquetado, más estricto que HTML, va a permitir una correcta interpretación de la información independientemente del dispositivo desde el que se accede a ella. Al contrario que HTML, XHTML puede incluir otros lenguajes tales como MathML, SMIL o SVG.

XHTML, al estar orientado al uso de un etiquetado correcto, exige una serie de requisitos básicos a cumplir en lo que a código se refiere. Entre estos requisitos se pueden mencionar:

- Los documentos deben estar *bien formados*, es decir, todos los elementos deben de tener etiquetas de cierre y estar anidados perfectamente.
- Los nombres de atributos y elementos deben ir en minúsculas, esto es muy importante ya que XML discrimina entre mayúsculas y minúsculas.
- Los elementos que no estén vacíos necesitan etiquetas de cierre.
- Los valores de los atributos deben ir entre comillas aunque sean numéricos.

4.4.3. CSS

CSS (*Cascade Syle Sheet*) es un mecanismo simple que describe cómo se va a mostrar un documento en pantalla. CSS se utiliza para dar estilo a documentos HTML y XML separando el contenido de la presentación. Los “estilos” definen cómo se van a mostrar en pantalla los documentos HTML y XML. CSS permite controlar los estilos y el formato de varias páginas Web al mismo tiempo, cualquier cambio en el estilo marcado para un elemento en la CSS afectará a todas las páginas vinculadas a esa CSS en las que aparezca dicho elemento.

CSS funciona mediante declaraciones sobre el estilo de uno o más elementos. Las hojas de estilo están compuestas por una o más de esas declaraciones aplicadas a un documento HTML o XML. La regla tiene 2 partes: el selector y la declaración; a su vez la declaración se divide en una propiedad y el valor que se le asigne.

```
h1 {color: red;}
```

El selector funciona como enlace entre el documento y el estilo; especifica que elementos se verán afectados por la declaración que lleve asociado. La declaración es la parte de la regla que establece cual será el efecto aplicado. En el ejemplo *h1* será el selector (en éste caso se aplica a la etiqueta HTML h1) y *color: red;* es el selector, que en este caso aplica al elemento indicado en el selector la propiedad *color* con valor rojo.

Existen 3 maneras de dar estilo a un documento: Asociando una hoja de estilo externa al documento mediante la etiqueta <*link*>, utilizando el elemento <*style*> dentro del documento al cual se quiere dar estilo y utilizando estilos directamente sobre el elemento que lo permita, a través del atributo *style*

4.4.3. JAVASCRIPT

Javascript es un lenguaje de programación que se utiliza principalmente para crear páginas Web dinámicas. Una página Web dinámica es aquella que incorpora efectos, acciones que se activan al pulsar botones y ventanas con mensajes de aviso al usuario.

Técnicamente Javascript es un lenguaje de programación interpretado, es decir, no es necesario compilar los programas para ejecutarlos. En otras palabras, los programas en Javascript se pueden probar en cualquier navegador sin necesidad de procesos intermedios.

Javascript fue diseñado de forma que se ejecutara en entornos muy limitados que permitiera a los usuarios confiar en la ejecución de de los scripts, de esta forma los scripts de Javascript no pueden comunicarse con recursos que no pertenezcan al mismo dominio desde el que se descargó el script. Entre otras limitaciones importantes, los scripts de Javascript no permiten cerrar ventanas que no hayan abierto esos mismos scripts; tampoco permiten el acceso a los archivos del ordenador del usuario, y tampoco pueden leer o modificar las preferencias del navegador utilizado.

Los navegadores actuales incluyen soporte para Javascript, no obstante existen diferencias en el dialecto utilizado ya que, mientras Microsoft Explorer utiliza JScript, el resto de navegadores (Firefox, Opera, Safari, Konqueror) utiliza Javascript.

4.4.4. DOM

Cuando surgió XML, surgió la necesidad de procesar y manipular el contenido de los archivos XML mediante los lenguajes de programación tradicionales. XML es un lenguaje sencillo de escribir pero complejo de procesar y manipular de forma eficiente. Por ello surgieron algunas técnicas entre las que se encuentra el DOM.

El DOM (*Document Object Model*) es un conjunto de utilidades especialmente diseñadas para manipular documentos XML. Por extensión DOM también se puede utilizar para manejar documentos XHTML y HTML. Técnicamente, DOM es una API de funciones que se pueden utilizar las páginas XHTML de forma más rápida y eficiente.

El árbol generado no sólo representa los contenidos del archivo original (mediante los nodos del árbol), sino que también representa sus relaciones (mediante las ramas del árbol que conectan los nodos).

Como ejemplo se considera la siguiente página HTML sencilla:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/
xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
       <head>
          <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
          <title>Página sencilla</title>
       </head>
       <body>
          <p>Esta página es <strong>muy sencilla</strong></p>
       </body>
</html>
```

Antes de poder utilizar las funciones DOM, los navegadores convierten automáticamente la página HTML anterior en la siguiente estructura de árbol de nodos.

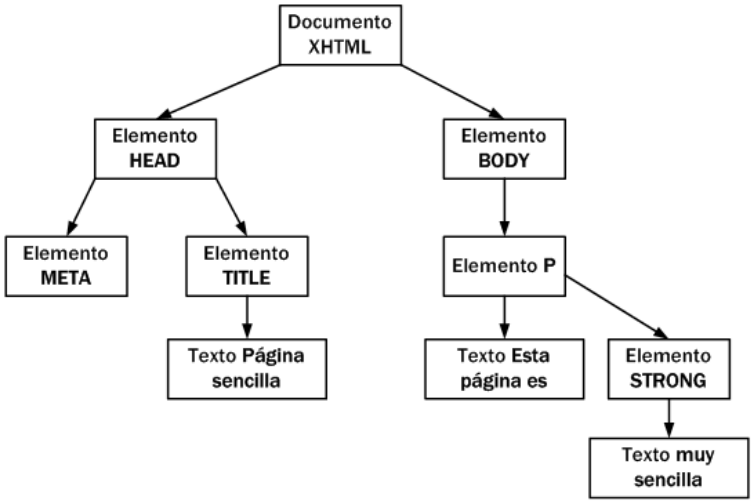


Figura 8: Árbol de nodos de un documento HTML

Aunque en ocasiones se asocia DOM con la programación Web y Javascript, la API (*Interfaz de Programación de Aplicaciones*) de DOM es independiente de cualquier lenguaje de programación, de hecho DOM está presente en la mayoría de los lenguajes de programación comúnmente empleados.

4.4.5. J2ME

J2ME (*Java 2 Micro Edition*) es una versión de Java enfocada a la aplicación de la tecnología Java en dispositivos electrónicos con capacidades computacionales y gráficas muy reducidas, tales como teléfonos móviles, PDA y electrodomésticos inteligentes. Los componentes que forman esta tecnología son:

- Una serie de máquinas virtuales Java con diferentes requisitos, cada una para un tipo de dispositivo.
- *Configuraciones*, que son un conjunto de clases básicas orientadas a conformar el corazón de las implementaciones para dispositivos de características específicas. Existen 2 configuraciones definidas en J2ME: *Connected Limited Device Configuration* (CLDC) enfocada a dispositivos con restricciones de procesamiento y memoria, y Connected Device Configuration (CDC) enfocada a dispositivos con más recursos.

— *Perfiles*, que son unas bibliotecas Java de clases específicas orientadas a implementar funcionalidades de más alto nivel para familias de dispositivos más específicas.

Un entorno de ejecución determinado de J2ME se compone de una selección de:

a) Máquina Virtual

b) Configuración

c) Perfil

d) Paquetes Opcionales

La parte móvil del sistema implementado está programada con configuración CDLC 1.1 y perfil MIDP 2.1.

Como ya se ha mencionado, CDLC está orientada a dispositivos dotados de conexión y con limitaciones en cuanto a capacidad gráfica, cómputo y memoria. Algunas restricciones vienen dadas por la máquina virtual utilizada, en este caso se trata de la KVM (*Kilo Virtual Machine*). La CDLC aporta las siguientes funcionalidades:

— Un subconjunto del lenguaje Java y todas las restricciones de su máquina virtual (KVM).

— Un subconjunto de las bibliotecas Java del núcleo.

— Soporte para E/S básica.

— Soporte para acceso a redes.

— Seguridad.

Esta configuración no se encarga del ciclo de vida de la aplicación, interfaces de usuario o manejo de eventos, sino que estas responsabilidades caen en manos de los *perfiles*.

Como ya se ha mencionado, el sistema implementado utilizará el perfil MIDP (*Mobile Information Device Profile*). Las aplicaciones programadas sobre este perfil reciben el nombre de MIDlet. Los estados de un MIDlet son:

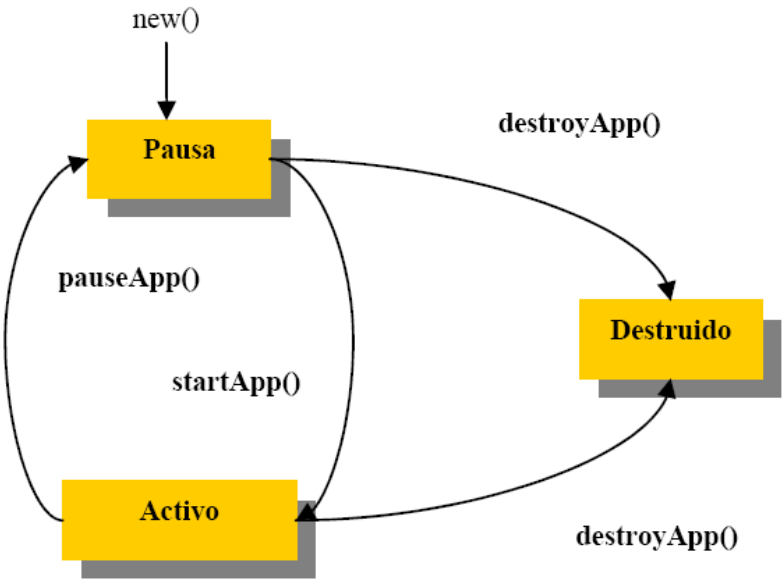


Figura 9: Diagrama de estados de un MIDlet

4.5. Tecnologías del lado del servidor

4.5.1. Servlets

Un servlet es una clase Java que se ejecuta en un servidor de aplicaciones para dar respuesta a una petición realizada desde un cliente. Básicamente un servlet recibe una petición de un usuario y devuelve un resultado a esa petición.

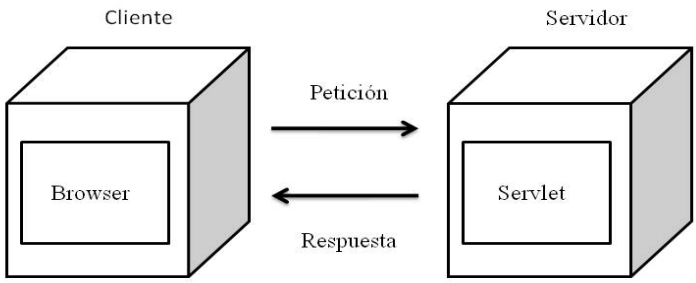


Figura 10: Comunicación cliente-servlet

Pueden existir diferentes tipos de servlets (HTTP, FTP, etc.), pero habitualmente se utilizan los servlets HTTP. Para este caso el cliente será un navegador Web que realiza una petición, el servlet recibirá la petición, realizará las operaciones pertinentes y devolverá una respuesta al cliente.

Entre las ventajas de usar servlets frente a otras tecnologías destacan la eficiencia, las utilidades para realizar las típicas tareas de un servidor (logging, gestión de sesiones y errores, cookies, etc.), la comunicación y ventajas heredadas del uso del lenguaje de programación Java (Gran número de APIS, portabilidad entre plataformas y servidores, seguridad y orientación a objetos).

El ciclo de vida de un servlet consiste en:

- Instanciación e inicialización: En la primera petición si no existen instancias del servlet, el contenedor Web carga la clase del servlet, crea una instancia y la inicializa llamando al método *init ().*
- Manejo de sucesivas peticiones: El contenedor Web crea un hilo y llama al método *service*; este método determina lo que llega en la petición y llama al método adecuado. Generalmente se atenderán peticiones GET y POST con lo cual los métodos a los que llamará *service* serán los métodos *doPost* y *doGet*, es por esto que sean estos dos métodos los que se implementen, dejando intacto el método *service.*
- Destrucción: Cuando el contenedor decide destruir el servlet, llama a su método *destroy*.

Se profundizará más sobre los servlets en el capítulo 5 de este proyecto.

4.5.2. JSP

JSP (*Java Server Pages*) es una tecnología basada en Java que permite simplificar el proceso de creación de sitios Web dinámicos utilizando dicho lenguaje de programación y una serie de etiquetas especiales determinadas. JSP es un lenguaje de script del lado del servidor.

Una página JSP va a ser un fichero con extensión .jsp, que mezcla código HTML con las etiquetas propias de este lenguaje. Una página JSP tendrá un aspecto muy similar a una página HTML, pero se transformará en clases de Java, que son servlets, para compilarse y generar los ficheros de clase correspondiente; esta operación se realizará siempre que se cargue una página por primera vez o varíe su contenido.

Dentro de una página JSP hay que diferenciar entre dos elementos, por un lado existen elementos que son procesados por el servidor, y por otro, código que el servidor ignora (normalmente código HTML). Cuando un cliente solicita una página JSP, se ejecutara dicha pagina devolviendo como resultado código HTML, este código es el resultado de la ejecución de la página JSP y comprende el contenido HTML contenido en la pagina y el resultado del contenido dinámico que ha sido ejecutado por el servidor.

Los elementos especiales de los que consta una página JSP son:

- Expresiones: *<% = expresión %>*, son evaluadas y el resultado se incluye a la salida.
- Scriptlets: *<% código %>*, bloques de código que se insertan en el método *_jspservice* (llamado por *service*).
- Declaraciones: *<%! código %>*, el código se inserta en el servlet fuera de los métodos existente. Es código de inicialización.

Otros elementos serían las *directivas*, las *acciones* y los *objetos implícitos*.

Se profundizará más sobre las páginas JSP en el capítulo 5 de este proyecto.

4.5.3. SQL

SQL (*Structured Query Lenguaje*) es un lenguaje estándar pensado para la consulta de bases de datos relacionales. Es un lenguaje normalizado que permitirá trabajar con cualquier tipo de lenguaje y cualquier tipo de base de datos.

Dentro del lenguaje SQL cabe diferenciar entre 2 conceptos:

- Lenguaje de Definición de Datos: Con sentencias para crear, alterar o borrar la base de datos. Las sentencias más importantes para este lenguaje serian:
  - CREATE DATABASE: Creación de la base de datos.
  - CREATE TABLE: Creación de cada una de las tablas de la base de datos.
  - DROP TABLE: Borrado de las tablas de la base de datos.
  - ALTER TABLE: Modifica la estructura de la tabla añadiendo o quitando atributos.

— Lenguaje de Manipulación de Datos: Con sentencias para consultar, insertar, actualizar y borrar instancias de la base de datos. Las sentencias principales son:

- SELECT: Selección las filas de una o varias tablas de la base de datos.
- INSERT: Añade filas a una tabla.
- UPDATE: Actualiza los valores de los atributos existentes en cada una de las filas de la tabla.
- DELETE: Elimina filas de una tabla.

Se profundizará más sobre el lenguaje SQL en el capítulo 5 de este proyecto.

4.5.4. Servidor Apache-Tomcat

Apache-Tomcat es un servidor Web con soporte para servlets y JSP escrito en Java, con lo cual puede ejecutarse en cualquier entorno que disponga de la maquina virtual de Java, y de libre distribución. Actualmente Apache-Tomcat es usado como servidor Web autónomo en entornos con alto nivel de tráfico y alta disponibilidad.

La jerarquía de directorios de Apache-Tomcat incluye:

— bin: Scripts de arranque y cierre.

— common: Clases comunes que pueden usar las aplicaciones Web.

— conf: Ficheros XML y sus correspondientes DTD para la configuración de Tomcat.

— logs: Logs del servidor y las aplicaciones Web.

— server: Clases utilizadas únicamente por el servidor Web.

— shared: Clases compartidas por todas las aplicaciones Web.

— webapps: Directorio que contiene las aplicaciones Web.

— work: Almacenamiento temporal de ficheros y directorios.

4.5.5. JDBC

JDBC (*Java Database Conectivity*) es una interfaz para trabajar con bases de datos relacionales que se divide en el driver para la conexión con la base de datos y el API para interactuar con la base de datos. Es un estándar que permite consultar de la misma forma SGBD de varios fabricantes.

Los pasos básicos para conectarse a una base de datos con JDBC se resumen en el siguiente esquema.

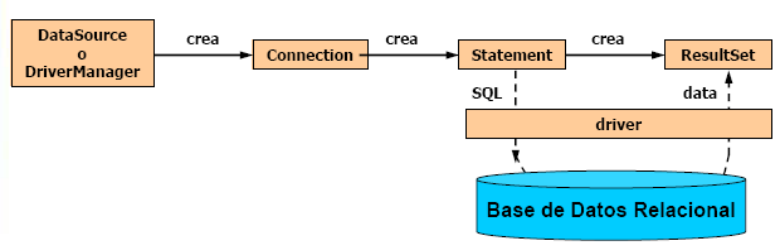


Figura 11: Pasos básicos para la conexión a bases de datos mediante JDBC

Se profundizará más JDBC en el capítulo 5 de este proyecto.

4.6. Otras tecnologías utilizadas

4.6.1. JavaMail API

JavaMail es una API opcional a la versión estándar de Java que proporciona funcionalidades de correo electrónico. Permite realizar desde tareas básicas como enviar, leer y componer mensajes, hasta tareas más sofisticadas como manejar gran variedad de formatos de mensajes y datos, y definir protocolos de acceso y transporte.

Aunque a primera vista pueda parecer que su utilidad se limita únicamente a la implementación de clientes de correo electrónico, su uso se puede generalizar a cualquier programa Java que necesite enviar y/o recibir mensajes, como es el caso del proyecto realizado.

JavaMail soporta por defecto los protocolos SMTP, POP3, IMAP, y las versiones seguras de estos: SMTPS, POP3S e IMAPS. El envió y recepción son independientes del protocolo, aunque se podrá necesitar un protocolo u otro en función de las necesidades.

Este API va unido al uso del paquete JAF (*JavaBeans Activation Framework*). Este paquete es el que se encarga de la manipulación de los canales (streams) de bytes de los tipos MIME, que pueden incluirse en un mensaje.

Las funciones más importantes de JavaMail son:

- Componer y enviar un mensaje: Es posible crear mensajes tanto de texto plano como mensajes que contengan adjuntos. Se pueden componer también mensajes que contengan lenguaje HTML, e incluso imágenes embebidas en el texto.
- Descargar mensajes: Se pueden descargar los mensajes desde la carpeta que se indique ubicada en el servidor que en ese momento estemos usando. Estos mensajes pueden contener texto plano, ficheros adjuntos, imágenes, etc.
- Usar flags para determinar si un mensaje ha sido leído, borrado, etc., en función de si estos flags están disponibles en el servidor.
- Establecer una conexión segura: En la actualidad, algunos servidores requieren una conexión segura ya sea para descargar el correo almacenado en ellos, como para enviar mensajes a través de ellos. JavaMail ofrece la posibilidad de establecer este tipo de conexión indicando que se trata de una conexión segura.
- Autenticarse en un servidor: Algunos servidores requieren autenticación ya no solo para leer sino para enviar. JavaMail ofrece la posibilidad de autenticación a la hora de enviar un correo.
- Definir protocolos: JavaMail soporta por defecto los protocolos de envío y recepción SMPT, IMAP y POP3, pero puede implementar otros protocolos.
- Manejar adjuntos: JavaMail ofrece la posibilidad de añadir archivos adjuntos a los mensajes de salida, así como obtener y manipular los adjuntos contenidos en un mensaje de entrada.

- Búsquedas: JavaMail permite realizar búsquedas de mensajes dentro de la cuenta correo en el propio servidor sin necesidad de descargar todo el correo.
- Acuse de recibo y prioridad: Se puede incluir acuse de recibo en los mensajes de salida para que el remitente sepa si un mensaje ha sido leído o no por un destinatario (no todos los servidores soportan esta funcionalidad). Se puede establecer la prioridad del mensaje.

4.6.2. Protocolo SMTP

SMTP (*Simple Mail Transfer Protocol*) es un conjunto de reglas que rigen el formato y la transferencia de datos en un envío de correo electrónico.

Los pasos básicos para realizar el envío de un correo electrónico son los siguientes:

- El autor del mensaje utiliza un cliente de correo electrónico para escribir un mensaje a un destinatario, el cliente de correo genera el código SMTP del mensaje y lo envía a un servidor SMTP saliente.
- El servidor SMTP saliente utiliza una consulta DNS para conocer los servidores de correo entrante del destinatario (llamados MX primario y MX secundario), y envía el correo SMTP al primero de ellos que responde (para que el receptor reciba el mensaje tiene que tener los MX configurados en su servidor de DNS de manera que apunten a los servidores SMTP entrantes).
- El servidor de destino almacena el mensaje de tal forma que el destinatario pueda acceder al él.

La comunicación SMTP se realiza sobre TCP, y normalmente en el puerto 25. Es una comunicación de tipo orden-respuesta delimitada por CLRF (retorno de carro + salto de línea). En esta comunicación se usarán las siguientes órdenes SMTP:

HELO	Abre una sesión con el servidor
MAIL FROM	Indica el autor del mensaje
RCTP TO	Indica los destinatarios del mensaje
DATA	Cuerpo del mensaje
.	Final del cuerpo del mensaje
QUIT	Cierra la sesión

Figura 12: Ordenes SMTP

Las respuestas que se recibirán serán un código y un texto donde el código será:

2XX	Orden realizada correctamente
3XX	Orden correcta recibida pero no requiere acción
4XX	Repetir orden correctamente
5XX	Error

Figura 13: Códigos de estado presentes en una comunicación SMTP

5. Análisis del sistema.

5.1. Dispositivo móvil.

Como se ha explicado anteriormente, la aplicación instalada en el dispositivo móvil, tendrá 2 funciones principales:

- Autenticación del usuario del teléfono móvil para un correcto acceso a las imágenes enviadas a través de Internet.
- Envío de la imagen resultante de una detección de movimiento a través de Internet.

5.1.1. Clase MenuInicio.java.

Esta clase es la que se encarga de comprobar si el número de teléfono corresponde al de un usuario registrado en la base de datos. Para ello existe un campo de texto en el cual introduciremos el teléfono móvil del usuario:

```
insertar = new TextField(9);
insertar.setConstraint(TextField.NUMERIC);
insertar.setInputModeOrder(new String[] { "123" });
```

Una vez introducido el número de teléfono pulsaremos el botón de *Si*, una vez hecho se obtendrá el número de teléfono introducido en el campo de texto mediante el método *getText()* y se comprobará si el número se ha introducido y que éste es correcto, es decir, si tiene 9 dígitos. Posteriormente se enviará el número de teléfono al servidor mediante una petición GET. Adicionalmente se guardará el número de teléfono introducido en la memoria del teléfono para recuperarlo a la hora de mandar la imagen.

```
String telefono = insertar.getText();
if (telefono != null && telefono.length() == 9) {
mandarTelefonoHttp(telefono);
guardarTelefono(telefono);
} else {
dialogoNumeroIncorrecto();
}
```

El método que se encarga de realizar el envío al servidor para su posterior validación es el método *mandarTelefonoHttp(String telf).* Este método recibe como parámetro una cadena de caracteres con el número de teléfono introducido en el campo de texto.

Para realizar una conexión a Internet mediante J2ME se usará la interfaz *HttpConexion* del paquete *javax.microedition.io.* En primer lugar hay que abrir la conexión con la URL a la que se quiera acceder.

```
String url = new String("http://localhost:8080/PFCWeb/MovilServlet");
hc = (HttpConnection) Connector.open(url);
```

Una vez abierta la conexión hay que especificar el método de la petición HTTP, también se añadirá el número de teléfono, en este caso como propiedad de la petición.

```
hc.setRequestMethod(HttpConnection.GET);
hc.setRequestProperty("telefono", telf);
```

En caso de producirse algún error al conectarse con el servidor se mostrará un mensaje de error y se procederá a cerrar la conexión.

```
} catch (IOException ioe) {
añadirError("No es posible realizar la conexion con el servidor,
compruebe que existe conexion a Internet e intentelo de nuevo. En caso
de persistir el problema continue con la aplicación sin conexión");
} finally {
try {
if (hc != null)
hc.close();
} catch (IOException ioe) {
}
```

Una vez realizada la petición, el servidor la procesará y devolverá una respuesta. En caso de que la autenticación no se haya realizado correctamente el servidor devolverá una respuesta 403 FORBIDDEN y se mostrará un mensaje explicativo. Por otro lado si se produce algún error en el servidor la respuesta será 500 INTERNAL SERVER ERROR, mostrándose asimismo un mensaje de error. Si el proceso se ha realizado exitosamente se recibirá una petición 200 OK, se pasará a la aplicación de detección de movimiento y se activará una variable booleana que servirá para, más tarde comprobar si el usuario ha dado su aprobación para el envío de imágenes al servidor a través de Internet.

```
if (hc.getResponseCode() == HttpConnection.HTTP_FORBIDDEN) {

añadirError("El numero de telefono introducido no corresponde con
ningun usuario registrado, introduzca un numero de telefono
correcto");

} else if (hc.getResponseCode() == HttpConnection.HTTP_INTERNAL_ERROR)
{
añadirError("El servidor esta fuera de servicio");

} else {

internet = true;
new MenuCaptura(midlet, conexionSms);

}
```

Una vez enviado el número de teléfono al servidor, se guardará en la memoria persistente del teléfono móvil gracias al RMS (*Record Management System*), que es un pequeño sistema de bases de datos que permite añadir información en la memoria no volátil del teléfono. El método que realiza esta función es *guardarTelefono(String telf).* Este método recibe como parámetro una cadena de caracteres con el número de teléfono introducido en el campo de texto.

Para poder almacenar un registro, en primer lugar se deberá abrir un *RecordStore*, una vez hecho esto ya se podría almacenar el registro que queramos, el dato a introducir deberá ser introducido como array de bytes. Realizadas estas dos operaciones se cerrará el *RecordStore*.

```
RecordStore telefono = RecordStore.openRecordStore("Telefono", true);
telefono.addRecord(telf.getBytes(), 0, telf.getBytes().length);
telefono.closeRecordStore();
```

5.1.2. Clase PantallaVideo.java.

Esta es la clase que se encargará de controlar la detección de movimiento y del envío de la imagen al servidor. Una vez se detecta un movimiento, se comprueba si el usuario ha aceptado el envío de imágenes a través de Internet, en cuyo caso se procederá a su envío:

```
if (deteccion.hayDeteccion()) {

if (MenuInicio.internet) {
mandarImagenHttp();

}
```

Al igual que en la clase *MenuInicio.java* explicada anteriormente, se debe abrir la conexión con la URL que deseemos. También se especificará el tipo de petición HTTP que se hace al servidor, en este caso POST, asimismo se añadirá el número de teléfono que se ha introducido al realizar la autenticación y que está guardado en la memoria no volátil del teléfono; el motivo de enviar éste parámetro es que al almacenar la imagen en el servidor, esta se guardará en una carpeta identificada por el número de teléfono que ha generado la imagen de tal forma que a la hora de acceder a ellas se diferencien claramente las imágenes mandadas desde teléfonos móviles distintos.

```
String url = new String("http://localhost:8080/PFCWeb/MovilServlet");
String numerotelf = obtenerNumero();
hc = (HttpConnection) Connector.open(url);
hc.setRequestMethod(HttpConnection.POST);
hc.setRequestProperty("telefono", numerotelf);
```

Una vez realizada la conexión se envía la imagen al servidor. En primer lugar la imagen se transformará en una cadena de caracteres usando codificación Base64.

```
String stringImage = Base64.encode(imageArray);
```

La codificación Base64 se basa en el uso de caracteres ASCII para codificar cualquier tipo de información mediante un código de 8 bits. Para realizar la codificación se ha hecho uso de una clase Java independiente realizada por *Stefan Haustein* en el año 2003.

Una vez se ha codificado la imagen se deberá incluir en el cuerpo de la petición, para ello se abrirá un *OutputStream* asociado a la conexión HTTP con el servidor y se escribirán los bytes correspondientes a la imagen codificada

```
out = hc.openOutputStream();
out.write(stringImage.getBytes());
```

En caso de producirse algún error tanto en la creación de la conexión como en la escritura en el *OutputStream*, ambas conexiones se cerrarán.

```
} catch (IOException ioe) {
ioe.printStackTrace();
} finally {
try {
if (out != null)
out.close();
if (hc != null)
hc.close();
} catch (IOException ioe) {

}
```

5.2. Cliente Web.

El cliente especificado para este proyecto será un navegador Web Mozilla Firefox. Dicho cliente tendrá la función de servir de interfaz entre el usuario y el servidor de tal manera que el usuario podrá registrarse en el sistema y acceder a la información enviada por su teléfono móvil. Para ello se realizarán una serie de peticiones el servidor que serán tratadas mediante código JavaScript y AJAX.

La primera vez que accedemos al servicio se nos muestra la página *index.jsp*, en esta página se mostrarán un formulario con dos campos de texto. El primero servirá para introducir el nombre de usuario y el segundo, la contraseña.

```
<form method="post" action="Login" name="formulariologin"
onSubmit="return valida(this)">
```

El campo *method* indicará el tipo de petición que se realizará al servidor, en este caso una petición POST. El segundo campo *action* específica que hacer al realizar la petición, en este caso la petición la tratará el servlet Login. El campo name identifica con un nombre el formulario para facilitar su acceso por parte del código JavaScript. El ultimo campo onSubmit es un evento JavaScript que indica que se ejecutará una función al darle al botón de enviar. La función que se ejecuta se encuentra en el archivo *Funciones.js*. La función *valida()* comprueba los campos del formulario y en caso de estar vacíos muestra una alerta indicando al usuario que debe rellenar el campo del formulario.

```
function valida(F) {

    if( vacio(F.login.value) == false ) {
            alert("Introduzca un nombre de usuario.");
            return false;
    }
    if( vacio(F.password.value) == false ) {
        alert("Introduzca un password.");
        return false;
}

    return true;


}
```

A esta función se le pasa el formulario como parámetro y se apoya en la función *vacio()* para comprobar que el valor almacenado en el campo del formulario no es nulo. En caso de no producirse este hecho, la función devolverá una variable booleana a false que le indica al formulario que no realice la petición al servidor.

Los campos del formulario se definen de la siguiente manera:

```
<input type="text" name="login" value="" maxlength="30"/>
<input type="password" name="password" value="" maxlength="10"/>
```

El campo *type* indica el tipo de caracteres que admite el campo de texto, para escribir el nombre de usuario se usará *text*, y para la contraseña se usará el tipo password cuya particularidad reside en que ocultará los caracteres de tal forma que no puedan ser leídos. El campo *value* indicará si se muestra algún carácter al iniciarse el formulario. El campo *maxlength* indica el máximo número de caracteres que se podrán escribir en el campo de texto.

El último elemento necesario será el botón mediante el cual enviaremos los datos al servidor.

```
<input id="btn" type="submit" value="Enviar"
onmouseOver="cambiarColor()" onmouseOut="restablecerColor()"/>
```

El campo *id* identifica con un nombre el botón. El campo *type* indica el tipo de botón, en este caso es un botón de tipo *submit*, es decir enviará datos. El campo *value* determina el texto que se mostrará en el botón. En último lugar tendremos 2 eventos JavaScript; *onmouseOver* es un evento que ejecuta una función cuando se pasa el

puntero del ratón por encima del botón, en este caso se ejecutará una función que cambia el color del botón. El evento *onmouseOut* ejecuta una función cuando el puntero del ratón abandona el botón, la función que se ejecuta en este caso devolverá al botón a su color original. Las funciones que se ejecutan para cambiar el color se encuentran en el archivo *Funciones.js.*

```
document.getElementById("btn").style.background="#F99403"
```

Para cambiar el color de un elemento utilizaremos DOM para obtener el elemento HTML sobre el cual se quiera actuar. Posteriormente se accederá a la etiqueta *style* lo cual nos permitirá cambiar la propiedad CSS que controla el color de fondo.

Por último cabe destacar la existencia de una función JavasScript que se ejecutará siempre que se cargue la página (evento *onLoad* del elemento *body*). Dicha función comprueba si se ha activado un flag de error utilizado en caso de que el registro de usuario no se haya realizado con éxito, en cuyo caso se muestra un mensaje de error.

```
function recuperaError(){

	var error = <%=request.getAttribute("Error")%>;
	if(error){

	document.getElementById("error").style.visibility="visible";
	}


}
```

La variable *error* se obtiene mediante código JSP. Este código obtiene una variable booleana enviada por el servlet de registro que indica si se ha producido el registro correctamente, en caso contrario modificaremos el estilo de un elemento HTML mediante DOM haciendo dicho elemento visible.

En caso de que sea la primera vez que accede al servicio se deberá realizar el registro del usuario en la base de datos. Para ello se muestra la página *registro.jsp*. Esta página contiene un formulario en el cual podremos introducir el nombre de usuario, contraseña, número de teléfono y correo electrónico. Como salvedad se pedirá que se confirme la contraseña con un campo de texto adicional.

La implementación de esta página no difiere en absoluto con la pagina *index.jsp* anteriormente explicada, con la salvedad de que al enviar un formulario se ejecuta la función *validaRegistro*, ubicada en el fichero *Funciones.js*

La función *validaRegistro* en una función JavaScript que se encarga de comprobar que los datos introducidos en el formulario son correctos. Dicha función recibe como parámetro el formulario. Esta función realiza tres tipos de comprobaciones, devolviendo una variable booleana *false* en caso de que se produzca algún error de validación, tal y como se hacía en el formulario de *index.jsp*. Las comprobaciones son las siguientes:

- Se comprobará si el formulario se ha rellenado, de la misma manera a como se realizaba en el formulario de *index.jsp*.
- Se comprueba si la contraseña introducida coincide con la introducida en el campo adicional de confirmación de contraseña, para ello se recoge el valor escrito en cada uno de los campos y se compara:

```
F.pass.value!=F.passconfirm.value
```

- Se comprueba que la dirección de correo electrónico es una dirección valida, para ello se ayuda de la función *mostrar_informacion* ubicada en el archivo Funciones.js. Esta función comprueba que la dirección de correo corresponde a *nombre@dominio.com* mediante una variable que almacena una expresión regular y el método *test* que compara la cadena que se le pasa por parámetro con el patrón de la expresión regular y devuelve un valor booleano true si la validación es correcta:

```
function mostrar_informacion(txt){
       var b=/^[^@\s]+@[^@\.\s]+(\.[^@\.\s]+)+$/;
       return b.test(txt)  ;
}
```

Una vez el usuario accede a la aplicación, se mostrará la página *aplicación.jsp* que será la encargada de mostrar la información demandada por el usuario, en este caso las fotos enviadas por el teléfono móvil y una pequeña explicación del funcionamiento del sistema.

Esta página tiene una serie de botones que, al pulsarlos, realizarán una petición AJAX al servidor pidiéndole una página JSP en función de que botón pulsemos.

```
<a  href="javascript:llamarasincrono('jsp/inicio.jsp',  'contenido');"
onmouseout="MM_swapImgRestore()"
onmouseover="MM_swapImage('btnMenu','','img/btnMenuover.png',1)">
```

Adicionalmente estos botones responden al evento JavaScript *onmouseover* y *onmouseout* cambiando la imagen que se mostrará en el botón a fin de mostrar al usuario gráficamente que se está situado encima del botón.

La función *llamarasincrono* será la encargada de realizar la petición AJAX. Esta función se encuentra en el fichero *aplicación.js*.

La función recibe dos parámetros: La página que se está pidiendo al servidor y el elemento HTML que contendrá dicha página.

En primer lugar, se debe instanciar el objeto *XMLHttpRequest*, que es el objeto clave para realizar peticiones con el servidor sin necesidad de tener que recargar la página, una de las razones por las que se ha decidido incluir esta tecnología en la implementación del sistema. En función del navegador, la implementación del objeto *XMLHttpRequest* varía, es por ello que se incluye una sencilla discriminación en función del navegador en el que se está ejecutando la función:

```
if (window.XMLHttpRequest){
    pagina_requerida = new XMLHttpRequest();
} else if (window.ActiveXObject){
    try{
        pagina_requerida = new ActiveXObject ("Msxml2.XMLHTTP");
    }catch (e){
    try{
      pagina_requerida = new ActiveXObject ("Microsoft.XMLHTTP");
    }catch (e){
    alert(e.message);
    }
}
```

En caso de usar un navegador basado en estándares tal como Mozilla Firefox, Opera o Internet Explorer 7.0/8.0, obtendremos el objeto *XMLHttpRequest*, en cambio si se utiliza el navegador Internet Explorer se deberá usar el objeto *ActiveXObject*, del tipo Msxml2.XMLHTTP para Internet Explorer 6.0, y Microsoft.XMLHTTP para versiones del navegador más antiguas.

Una vez se obtiene la instancia del objeto *XMLHttpRequest*, se prepara la función encargada de procesar la respuesta del servidor a la petición realizada. Para ello se lee la propiedad *onreadystatechange* del objeto *XMLHttpRequest*, la cual permite asignar la función que procesará la respuesta, en este caso la función *cargarpagina()* ubicada en *aplicación.js*.

La función *cargarpagina()* recibe como parámetros la instancia del objeto *XMLHttpRequest* obtenida y el identificador del elemento que contendrá la página solicitada. A continuación se comprueba en primer lugar si se ha recibido respuesta del servidor, en caso afirmativo se comprueba si la respuesta ha sido correcta:

```
if (pagina_requerida.readyState == 4 && (pagina_requerida.status ==
200 || window.location.href.indexOf ("http") == - 1))
```

Una vez realizadas estas comprobaciones se usa la propiedad *innerHTML* del DOM para incluir el contenido de la respuesta del servidor en el contenedor que la albergará. El contenido de la respuesta del servidor se obtiene mediante la propiedad *responseText*.

```
document.getElementById(id_contenedor).innerHTML=
pagina_requerida.responseText;
```

Por último volviendo a la función *llamarasincrono()*, se debe realizar la petición HTTP al servidor:

```
pagina_requerida.open ('GET', url, true);
pagina_requerida.send (null);
```

La primera línea de código crea una petición GET sin parámetros solicitando la URL especificada por el parámetro *url*; el tercer parámetro indica que se desea realizar una petición asíncrona. Para ello se utiliza el método *open()* del objeto *XMLHttpRequest*.

Una vez creada la petición se envía mediante el método *send()* del objeto *XMLHttpRequest*. El parámetro que se le pasa a la función indica el contenido que se envía al servidor que, en este caso, es nulo ya que no se necesita enviar ningún dato.

5.3. Servidor.

El servidor tiene la función principal de recibir las peticiones realizadas tanto por el cliente Web como por el teléfono móvil, procesar la petición y enviar la respuesta al cliente que corresponda en cada caso. Está implementado mediante servlets, existiendo 2 tipos: Servlet que recoge peticiones realizadas desde un cliente Web y servlet que recoge peticiones realizadas desde el teléfono móvil.

5.3.1. Clase MovilServlet.java.

Esta clase se encarga de procesar las peticiones realizadas desde el teléfono móvil. En este caso se realizarán 2 tipos de peticiones:

- Envío del número de teléfono móvil para comprobar si corresponde con el de un usuario registrado. El método de la petición HTTP realizada es GET.
- Envío de la imagen resultante de una detección de movimiento. La imagen se enviará mediante una petición POST.

Para procesar las peticiones, la implementación de los servlets dispone de 2 métodos: *doGet(HttpServletRequest request, HttpServletResponse reponse)* para peticiones GET, y *doPost(HttpServletRequest HttpServletResponse request, response)* para peticiones POST.

El método *doGet*, en primer lugar obtendrá de la cabecera de la petición el número de teléfono que se ha introducido en el formulario correspondiente de la aplicación instalada en dicho dispositivo.

```
String telefonoMovil = request.getHeader("telefono");
```

Este número de teléfono deberá coincidir con el número de teléfono de un usuario registrado en la base de datos, para ello existe la clase *BaseDatos.java* que proporciona los métodos necesarios para el acceso a la base de datos (Esta clase se explicará posteriormente).

```
BaseDatos bbdd = new BaseDatos();
```

Para comprobar si el número de teléfono existe en la base de datos se hará uso de la clase *BaseDatos.java*. En primer lugar se abre la conexión con la base de datos, posteriormente se almacena en una variable booleana si el número existe. Por último cerramos la conexión con la base de datos. Si la comprobación es invalida, la variable booleana almacena el valor a false y se enviará una respuesta 403 FORBIDDEN al cliente. En caso de producirse algún error en este proceso se enviará al cliente una respuesta 500 INTERNAL SERVER ERROR.

```
try {
bbdd.abrirConexion();

boolean existe = bbdd.esMovil(telefonoMovil);

bbdd.cerrarConexion();

if(!existe){
response.sendError(HttpServletResponse.SC_FORBIDDEN, "El telefono no
existe en la base de datos");
}
} catch (Exception e) {
      response.sendError(HttpServletResponse.SC_INTERNAL_SERVER_ERROR,
"Error en la base de datos");
e.printStackTrace();
}
```

Por otro lado el método *doPost* recibirá el número de teléfono y la imagen enviada desde el teléfono móvil y almacenará la imagen en el sistema de archivos del servidor. Adicionalmente almacenará en la base de datos información relativa a la imagen tal como la fecha de recepción, la ruta donde se almacenará la imagen y el número de teléfono que la ha generado.

Para realizar este proceso, recogeremos el número de teléfono de la cabecera de la petición así como la imagen enviada, en formato *InputStream*.

```
String telefono = request.getHeader("telefono");
InputStream in = request.getInputStream();
String url = guardarImagen(in, telefono);
```

El método *guardarImagen(InputStream in, String telefono)* será el encargado de crear el sistema de archivos en el servidor donde se guardará la imagen, devolviendo la ruta donde se ha guardado dicha imagen.

Para crear un nuevo directorio se hará uso de la clase *File*.

```
File directory = new
File("C:/Users/JUAN/j2eeprojects/PFCWeb/WebContent/imgmvl/"+telefono);
directory.mkdir();
```

Para cada usuario de la aplicación instalada en el teléfono móvil se creará un nuevo directorio identificado por el número de teléfono que se ha introducido en el formulario de autenticación de la aplicación, de tal forma que el acceso posterior a las imágenes sea más eficiente y libre de errores.

Cada imagen almacenada estará identificada por un número. Dicho número se incrementa por cada vez que el servidor recibe una petición POST, es decir, cada vez que se guarde una imagen, independientemente del usuario que la haya generado.

```
String path =
"C:/Users/JUAN/j2eeprojects/PFCWeb/WebContent/imgmvl/"+telefono+"/imag
e"+imagenes+".jpg";
```

Una vez creado el sistema de archivos en el servidor, se procederá a la recuperación de la información contenida en la petición, en este caso la imagen codificada en Base64.

```
byte[] data = Base64.decode(s);
```

Una vez obtenida y decodificada la imagen, se escribirá en el sistema de archivos del servidor que corresponda y se devolverá la ruta donde se ha guardado.

```
f1 = new FileOutputStream(path);
f1.write(data);
f1.close();
return path;
```

A continuación se inserta en la base de datos información relativa a la imagen que se acaba de guardar en el sistema de archivos del servidor. La información que se guardará será la ruta donde se ha almacenado la imagen (para luego poder extraer y visualizar la imagen), y el número de teléfono móvil que ha generado la imagen.

```
insertarImagen(telefono, url);
```

La última función que realizará esta clase será la de enviar un mensaje de correo electrónico al correo electrónico del usuario. La clase *EnviarMail.java*, que se explicará más detalladamente a lo largo de este capítulo, será la encargada de generar el mensaje, conectarse al servidor de correo electrónico y enviar la información. El método *obtenerCorreoElectronico* obtendrá de la base de datos la dirección de correo electrónico asociada al número de teléfono que ha mandado la imagen, se recuerda que para poder hacer uso de esta función del sistema, el usuario ha debido identificarse en la aplicación móvil.

```
EnviarMail correo = new EnviarMail(obtenerCorreoElectronico(telefono),
url);
```

5.3.2. Clase Registro.java.

Esta clase se encarga de añadir un nuevo usuario a la base de datos a partir de unos datos introducidos en un formulario HTML y enviados al servidor mediante una petición POST.

Se implementará únicamente el método *doPost*, que recogerá los parámetros introducidos en el formulario. Los datos de registro serán: nombre de usuario, contraseña, número de teléfono y correo electrónico.

```
usuario = request.getParameter("user");
contrasena = request.getParameter("pass");
telefono = request.getParameter("telefono");
mail = request.getParameter("mail");
```

Si el proceso de añadir un usuario a la base de datos se realiza correctamente, se delegará la petición a una página Web de éxito, en caso contrario se delegará a una de error.

```
try {
añadirUsuario();
url="/jsp/exito.jsp";
} catch (Exception e) {
url="/jsp/error.jsp";
```

```
e.printStackTrace();
}
```

Para delegar una petición a cualquier otro recurso, tanto JSP, HTML, etc. Se hará uso de la interfaz *RequestDispatcher*, más concretamente del método forward, que delega el procesamiento de la petición al recurso especificado en el argumento del objeto *RequestDispatcher* sobre el que se aplica.

```
rd=request.getRequestDispatcher(url);
rd.forward(request, response);
```

5.3.3. Clase Login.java.

Esta clase comprueba si el usuario está registrado en la base de datos. La autenticación se realiza a partir del nombre de usuario y contraseña que el usuario ha introducido en un formulario HTML y enviado al servidor mediante una petición POST.

Esta clase recoge los datos enviados por el usuario en el método *doPost*.

```
user=request.getParameter("login");
pass=request.getParameter("password");
```

Posteriormente se comprobará si los datos son correctos, en cuyo caso se obtendrá de la base de datos el número de teléfono del usuario para posteriormente incluirlo en la información de la sesión y se indicará la página JSP a la cual se delegará la petición. En caso de que la autenticación no se realice con éxito, se delegará la petición a la página de inicio y se añadirá un parámetro a la petición que servirá para que se muestre un mensaje que indique que la autenticación no se ha realizado exitosamente.

```
if(bbdd.esUsuario(user, pass)){
telefono = bbdd.obtenerTelefono(user);
url=response.encodeURL("jsp/aplicacion.jsp");
}else{
url="index.jsp";
error=true;
request.setAttribute("Error", error);
}
```

Una vez realizado este paso, obtendremos las imágenes guardadas en el servidor para el usuario registrado.

```
imagenes=bbdd.recuperarImagen(telefono);
```

En caso de producirse algún error, se redireccionará al usuario a una página de error.

```
} catch (Exception e) {
url="jsp/error.jsp";
e.printStackTrace();
}
```

Una vez realizada la autenticación con éxito se deberá crear una sesión para el usuario. Para ello se utiliza la interfaz *HttpSession*, más concretamente el método *getSession*, que crea una nueva sesión.

```
sesionactual = request.getSession(true);
```

A la sesión se añadirá una serie de información, tal como el nombre del usuario, el número de teléfono y las fotos que en ese momento están guardadas en el servidor.

```
sesionactual.setAttribute("Usuario", user);
sesionactual.setAttribute("Telefono", telefono);
sesionactual.setAttribute("Fotos", imagenes);
```

Por último de delegará la petición tal como se ha explicado en el apartado anterior.

5.3.4. Clase Foto.java.

Esta clase describe la foto que se ha guardado en el servidor. Para ello se definen 3 parámetros:

- *String teléfono*: Número de teléfono que ha enviado la imagen.
- *String url*: Ruta donde se ha almacenado la imagen en el servidor.
- *String fecha*: Fecha de obtención de la foto.

Esta clase incluye además los métodos para acceder y modificar los parámetros descritos.

```
public String getTelefono() {
		return this.telefono;
	}

	public void setUrl(String url) {
		this.url=url;
	}

	public String getUrl() {
		return this.url;
	}

	public void setFecha(String fecha)
{
		this.fecha=fecha;
	}
	public String getFecha() {
		return this.fecha;
	}

}
```

5.3.5. Clase EnviarMail.java.

Esta clase es la encargada de implementar el envío de correos electrónicos en el sistema. Sus dos funciones principales serán la creación del correo electrónico con la imagen que se enviará, y el envío propiamente dicho. Esta clase está implementada mediante las API´s de Java encargadas del envío recepción y almacenamiento de correos electrónicos: JavaMail 1.4.2 y Jaf 1.0.2.

Lo primero que se implementa en esta clase será la sesión de correo a utilizar, para ello JavaMail nos proporciona el objeto *Session* que define una sesión de correo básica. Este objeto se aprovecha del objeto *Properties* del paquete *java.util* para obtener información que se pueda compartir en toda la aplicación.

La información que se ha decidido incluir en este caso es:

— Servidor SMTP a través del cual se enviará el correo electrónico. En este caso se ha elegido un servidor de correo externo tal y como es el servidor Gmail.

```
props.setProperty("mail.smtp.host", "smtp.gmail.com");
```

— Se habilita el comando STARTTLS para poder cambiar a una conexión segura mediante el protocolo TLS antes de iniciar un envío de información.

```
props.setProperty("mail.smtp.starttls.enable", "true");
```

— Puerto en el que está escuchando el servidor de correo electrónico. Por definición el servidor Gmail escucha el puerto 587.

```
props.setProperty("mail.smtp.port", "587");
```

— Usuario de la cuenta de correo electrónico a través de la cual se envía el mensaje.

```
props.setProperty("mail.smtp.user", "juanirulillo@gmail.com");
```

— Indicación de que será necesaria la autenticación para poder utilizar el servicio de correo electrónico.

```
props.setProperty("mail.smtp.auth", "true");
```

Una vez definidas las propiedades se podrá obtener una instancia del objeto *Session* asociándole las propiedades especificadas anteriormente mediante el método

getDefaultInstance(). Mediante este método se crea la sesión por defecto, y esta sesión podrá ser compartida cuantas veces se quiera.

```
Session session = Session.getDefaultInstance(props,null);
```

Una vez realizados estos pasos se pasará a crear el mensaje de correo electrónico así como su contenido. Para ello JavaMail nos proporciona la clase *MimeMessage*, la cual proporciona soporte para el envío de mensajes de correo que entienden tipos MIME. MIME (Multipurpose Internet Mail Extensions) es un estándar que clasifica recursos y provee a los programas de información útil para poder manejar dichos recursos. En este caso se enviarán dos tipos MIME, “text/plain” y “text/html”, es decir texto plano y texto en lenguaje HTML. Así mismo se asociará el origen, el destino y el asunto del mensaje mediante métodos propios de la clase *MimeMessage*:

```
MimeMessage message = new MimeMessage(session);
message.setFrom(new InternetAddress("juanirulillo@gmail.com"));
message.setSubject("Detectado Intruso");
message.addRecipient(Message.RecipientType.TO,new InternetAddress(to));
```

El método *setFrom()* especifica el destinatario. El parámetro que se le pasa será un objeto de la clase *InternetAddress*, esta clase representa una dirección de Internet según la especificación RFC822, típicamente user@host.domain. El método *addRecipient()* especifica la dirección del destino del mensaje, dicha dirección será pasada por parámetros en el constructor de la clase. La manera de especificar la dirección es idéntica al método explicado anteriormente. Por último el método *setSubject()* añade el asunto del correo electrónico.

A partir de aquí el mensaje de correo electrónico está creado, ahora solo falta crear el contenido del mismo: Para ellos JavaMail nos proporciona la clase *Multipart*, que actúa de contenedor para las partes que tenga el mensaje. En concreto se usa la subclase *MimeMultipart* que no es más que una implementación de la clase *Multipart* para poder trabajar con tipos MIME.

En primer lugar se creará una parte del mensaje en texto plano para poder recurrir a ella en caso de que el envío de la parte con texto HTML y, por ende con la imagen, falle.

```
Multipart multipart = new MimeMultipart("alternative");
```

El parámetro *alternative* indica que la información en esta parte del mensaje es relativamente secundaria respecto al contenido principal.

Para esta parte del mensaje se creará y añadirá el contenido mediante el objeto *MimeBodyPart* y su método *setContent()*.

```
MimeBodyPart thisBodyPart;
String outText = "Se ha detectado un intruso";
thisBodyPart = new MimeBodyPart();

thisBodyPart.setContent(outText,"text/plain");
```

A la hora de añadir el contenido se le indicará al método el tipo MIME que alberga, en este caso texto plano. Por último se añade esta parte del mensaje al mensaje principal.

```
multipart.addBodyPart(thisBodyPart);
```

A continuación se creará otro objeto *MimeMultipart* para albergar el contenido del mensaje en lenguaje HTML e incluir la imagen a enviar.

```
Multipart htmlMultipart = new MimeMultipart("related");
```

El parámetro *related* indica que el contenido que se añada a esta parte del mensaje será considerado el principal.

A continuación se añade el texto en HTML que se mostrará en el mensaje, para ello se recurre de nuevo al objeto *MimeBodyPart*, su método *setContent()* para agregar contenido y *addBodyPart* para añadir todo esto al mensaje principal.

```
String outHTML ="<h1>Se ha detectado un intruso</h1>";
thisBodyPart.setContent(outHTML, "text/html");

htmlMultipart.addBodyPart(thisBodyPart);
```

Como particularidad se puede observar que el texto a insertar está escrito en formato HTML y así se le indica al método *setContent()*.

Queda por tanto añadir la imagen a partir de una ruta obtenida de la base de datos y pasada al constructor de la clase mediante un parámetro, y un identificador único de la imagen. Se volverá a hacer uso del objeto *MimeBodyPart* para insertar la

imagen. Para obtener la imagen a enviar deberemos hacer uso de la interface *DataSource* del paquete *javax.activation* (correspondiente a la API Jaf 1.0.2). Esta interface encapsula y abstrae al programador del proceso de inclusión de la imagen en el mensaje. Así mismo se dotará a la imagen de un nombre y un identificador con tal de ser identificada en el destino.

```
DataSource source = new FileDataSource(ruta);
thisBodyPart.setDataHandler(new DataHandler(source));
thisBodyPart.setFileName(identificador);
thisBodyPart.setHeader("Content-ID","<"+identificador+">");
```

Para poder enviar todo lo creado se deben asociar todos los objetos *MimeMultipart* creados hasta ahora al objeto *MimeMessage* creado inicialmente.

```
message.setContent(multipart);
```

El mensaje ya ha sido creado, por lo tanto se debe indicar al sistema que lo mande al destinatario elegido. Para ello JavaMail proporciona la clase *Transport*. Esta clase habla el lenguaje específico del protocolo a utilizar para el envío.

```
Transport t = session.getTransport("smtp");
t.connect("smtp.gmail.com", "juanirulillo@gmail.com",
"tarifaplanaya");
t.sendMessage(message, message.getAllRecipients());
t.close();
```

El método *getTransport()* especifica el tipo de protocolo que se utilizará para el envío, en este caso protocolo SMTP. El método *connect()* conectará el sistema al servidor de correo electrónico, en este caso Gmail. Tal y como se había indicado en las propiedades se requiere autenticación, es por ellos que se añade el nombre de correo y la contraseña para acceder al servidor de correo.

El método *sendMessage()* es el encargado de enviar el mensaje. A él se le añade el mensaje creado anteriormente y todos los destinatarios a los que vaya dirigido, en este caso será un solo destinatario, pero está implementado pensando que en un futuro este mensaje pueda ser enviado a diferentes usuarios del sistema.

El último paso en la creación y envío de mensajes de correo electrónico será cerrar la conexión con el servidor de correo mediante el método *close()*.

5.3.6. Base de datos.

La base de datos es una de las partes críticas del sistema ya que asegurará que ningún usuario pueda acceder al servicio sin haberse registrado previamente. Asimismo almacenará información relativa a las imágenes mandadas por el teléfono móvil tales como el número de teléfono que la ha generado, la ruta donde se ha almacenado la imagen en el servidor y la fecha en la que se ha almacenado la imagen.

La base de datos es del tipo MySQL y consta de 2 tablas.

Figura 14: Base de datos del sistema

La tabla *usuario* describe a un posible usuario de la aplicación. Los campos de los que consta esta tabla son:

- *LOGIN:* Este campo de la tabla almacena el nombre de usuario. El tipo de dato que almacena es VARCHAR, es decir, una cadena de caracteres de longitud variable; en este caso la máxima longitud de la cadena es de 30 caracteres.
- *PASSWORD:* Este campo de la tabla almacena la contraseña del usuario para acceder al sistema. El tipo de dato que almacena es VARCHAR de longitud máxima 10 caracteres.

— *TELEFONO:* Este campo almacena el número de teléfono del usuario. Este campo será útil para autenticar la aplicación instalada en el teléfono móvil. El tipo de dato que almacena es VARCHAR, de longitud máxima 9 caracteres, tantos como dígitos tiene un número de teléfono (sin contar el código del país).

— *CORREO:* Almacena la dirección de correo electrónico especificada por el usuario a la cual desea que se manden los mensajes de correo electrónico que el sistema envía cuando se detecta movimiento en la aplicación móvil.

Por otra parte la tabla *imágenes* describe una imagen mandada desde el teléfono móvil y guardada en el servidor. Los campos de los que consta esta tabla son:

— *TELEFONO:* Este campo de la tabla almacena el número de teléfono que ha enviado la imagen al servidor. El tipo de dato que almacena es VARCHAR, de longitud máxima 9 caracteres, tantos como dígitos tiene un número de teléfono (sin contar el código del país).

— *URL:* Almacena la ruta en la que se ha guardado la imagen en el servidor. Gracias a este campo, el usuario podrá recuperar la imagen y visualizarla en un navegador Web. El tipo de dato que almacena es VARCHAR.

— *FECHA:* Almacena la fecha en la que la imagen ha llegado y se ha guardado en el servidor. El tipo de dato almacenado es TIMESTAMP, este tipo de datos representa la fecha actual en formato YYYY-MM-DD HH:MM:SS.

Los campos TELEFONO tanto de la tabla *usuario* como de la tabla *imágenes* están relacionados mediante una FOREIGN KEY, que es un tipo de restricción consistente en que el contenido de una tabla hija será uno de los valores existentes en un campo de la tabla padre, es decir, siempre que se añada un campo en la tabla *imágenes*, el campo TELEFONO deberá tener el mismo valor que el campo del mismo nombre en la tabla *usuario*. Si se intenta insertar una imagen con un número de teléfono no existente en la tabla *usuario*, la tabla dará un error. Para asegurar la integridad referencial de las 2 tablas se ha impuesto la condición ON DELETE CASCADE, es

decir, si se borra un usuario de la base de datos, todas las imágenes asociadas a ese usuario también se eliminarán.

Las clases JAVA que manejan el acceso a la base de datos son *BaseDatos.java* y *Query.java*. Estas 2 clases usan JDBC (*Java Database Connectivity*).

- Clase Query.java

Esta clase es la encargada de proporcionar los métodos para poder realizar peticiones a la base de datos. En primer lugar se deberá crear un objeto de la clase *Statement* asociado a la conexión actual a la base de datos gracias al método *createStatement()*. Esto se realiza en el constructor de la clase *Query*.

```
public Query(Connection con) throws SQLException {
        stmt=con.createStatement();
    }
```

Los métodos implementados para realizar peticiones a la base de datos son:

— *doUpdate (String query)*: Este método recibe como parámetro la sentencia SQL que se quiera ejecutar. Se usará dicho método para realizar sentencias SQL del tipo INSERT (Insertar una fila en una tabla de la base de datos), UPDATE (Actualizar un valor de una fila en una tabla de la base de datos) y DELETE (Para borrar una fila de la tabla). Esto se consigue gracias al método de la clase *Statement executeUpdate*. Este método devuelve un número con el número de filas insertadas. Borradas o actualizadas.

```
public int doUpdate(String query) throws SQLException {
        return stmt.executeUpdate(query);
}
```

— *doSelect (String query)*: Al igual que el método anterior, éste también recibe como parámetro la sentencia SQL que se quiera ejecutar. En éste caso el método será utilizado para realizar peticiones del tipo SELECT (Seleccionar una fila o un campo de una tabla de la base de

datos). Para ello la clase *Statement* proporciona el método *executeQuery*. Este método devolverá los resultados obtenidos almacenados en un objeto *ResultSet*, que no es más que una tabla con un cursor.

```
public ResultSet doSelect(String query) throws Exception {
        ResultSet rs = stmt.executeQuery(query);
        return rs;

    }
```

El último método proporcionado será el encargado de eliminar el objeto de la clase *Statement*.

```
  public void close() throws SQLException {
        stmt.close();

     }
```

- Clase BaseDatos.java

Esta clase se encarga de implementar los métodos para conectarse a la base de datos y añadir u obtener información. Los métodos de los que consta esta clase son:

— *abrirConexión()*: Este método será el encargado de abrir la conexión con la base de datos. En primer lugar se debe cargar el driver JDBC que conectará la aplicación con dicha base de datos.

```
Class.forName("com.mysql.jdbc.Driver");
```

Posteriormente hay que crear la conexión con la base de datos mediante el objeto *Connection* proporcionado por la API de JDBC. Para crear la conexión se hará uso del método *getConnection*, el cual recibe como parámetros la URL donde está ubicada la base de datos, su login y su contraseña.

```
conexion=DriverManager.getConnection(
"jdbc:mysql://localhost:3306/PFCWEB", "root", "" );
```

Por último se inicializará un objeto de la clase Query explicada anteriormente para la conexión creada anteriormente.

```
consulta = new Query(conexion);
```

— *cerrarConexion()*: Este método cierra la conexión con la base de datos haciendo uso del método *close()* de la clase *Connection*.

— *esUsuario(String login, String pass)*: Este método comprobará en la base de datos si el usuario y la contraseña introducidos por parámetros existe. La sentencia SQL que permitirá obtener los datos necesarios para la comprobación será:

```
String query="SELECT LOGIN, PASSWORD FROM USUARIO
WHERE LOGIN='"+login+"'AND PASSWORD='"+pass+"';";
```

Esta sentencia equivaldría a seleccionar lo que esté almacenado en los campos LOGIN y PASSWORD de la tabla USUARIO siempre y cuando coincida con los datos pasados por parámetros.

Esta consulta dará lugar a un objeto *ResultSet* donde se almacenarán los registros resultantes de la consulta. Para realizar la comprobación recorremos este objeto hasta que no existan más elementos.

```
if(resultados.next()){
                return true;
        }
        else{
                return false;

        }
}
```

En caso de que la comprobación se haya realizado con éxito, éste método devolverá un valor booleano a true.

— *añadirUsuario(String login, String password, String telefono, String mail)*: Este método será el encargado de insertar un nuevo usuario en la base de datos una vez cumplimentado el formulario de registro. La sentencia SQL que lo hace posible será:

```
query="INSERT INTO USUARIO (LOGIN, PASSWORD, TELEFONO, CORREO)
VALUES('"+login+"','"+password+"','"+telefono+"','"+mail+"');";
```

Esta consulta equivaldría a insertar en la tabla USUARIO los campos LOGIN, PASSWORD, TELEFONO, CORREO, con los valores pasados por parámetros al método.

— *obtenerTelefono(String user)*: Este será el método utilizado para obtener el número de teléfono asociado a un usuario que se le pasará al método mediante un parámetro de tipo *string*. La sentencia SQL será:

```
query="SELECT TELEFONO FROM USUARIO WHERE LOGIN ='"+user+"';";
```

Esta consulta almacenará en un objeto de la clase ResultSet todos los números de telefono que coincidan con el nombre de usuario pasado por parámetro. Una vez obtenidos estos resultados, recorreremos el objeto y se obtendrá el número de teléfono almacenándolo en una variable de tipo *string*.

```
while(resultados.next()){
      telefono=resultados.getString("TELEFONO");
      return telefono;
}
```

Para obtener un objeto *string* almacenado en un *ResultSet* se recurrirá al método *getString()*, al cual se le pasa por parámetro el nombre del campo de la tabla al cual se quiera acceder.

— *obtenerCorreo(String telf)*: Este método obtiene el correo electrónico asociado a un número de teléfono de un usuario registrado en la base de datos. La implementación de este método es idéntica a la del método *obtenerTelefono()* explicado anteriormente. La única diferencia es que se le pasa por parámetro el número de teléfono del usuario del cual queramos obtener su dirección de correo electrónico.

— *esMovil(String telefono)*: Comprueba si el número de teléfono de usuario pasado por parámetros se encuentra registrado en la base de datos. La implementación de este método es idéntica a la del método *esUsuario()*.

— *añadirImagen(String telf, String path)*: Este método se encargará de insertar la información relativa a una imagen procedente del teléfono móvil, y guardada en el servidor en la base de datos. La sentencia SQL que realiza esta acción será:

```
query="INSERT INTO IMAGENES (TELEFONO, URL,
FECHA)VALUES('"+telf+"','"+path+"',now());";
```

La sentencia inserta en la tabla IMÁGENES los campos TELEFONO y URL pasados por parámetros, donde *url* será la ruta relativa donde estará guardada la imagen en el servidor. El valor del campo FECHA viene determinado por la función *now()*, que devuelve la fecha y hora actual.

— *recuperarImagen(String telefono)*: Con este método se obtendrá la información relativa a una imagen guardada en el servidor, y enviada por un número de teléfono cuyo valor se pasa por parámetro. La sentencia SQL utilizada es la siguiente:

```
String query="SELECT * FROM IMAGENES WHERE
TELEFONO='"+telefono+"' ORDER BY FECHA DESC;";
```

De esta manera se seleccionará todo lo que exista en la tabla IMÁGENES cuyos valores coincidan con el número de teléfono pasado por parámetro. Adicionalmente se ordenarán los resultados por fecha colocando en primer lugar la última imagen que se haya almacenado en el servidor.

Los resultados obtenidos se guardarán en un objeto de la clase *ArrayList<Foto>*, dicho *ArrayList* está particularizado para almacenar elementos de la clase *Foto*, en caso contrario se producirá una excepción.

```
while(resultados.next()){

String telf = resultados.getString("TELEFONO");
String url = resultados.getString("URL");
String fecha = resultados.getString("FECHA");
Foto foto = new Foto(telf, url, fecha);
fotos.add(foto);

}
return fotos;
```

6. Funcionamiento del sistema.

Para iniciar la aplicación Web se deberá abrir un navegador y acceder a la URL http://localhost:8080/PFCWeb. Se abrirá la página de inicio en la que el usuario deberá autenticarse.

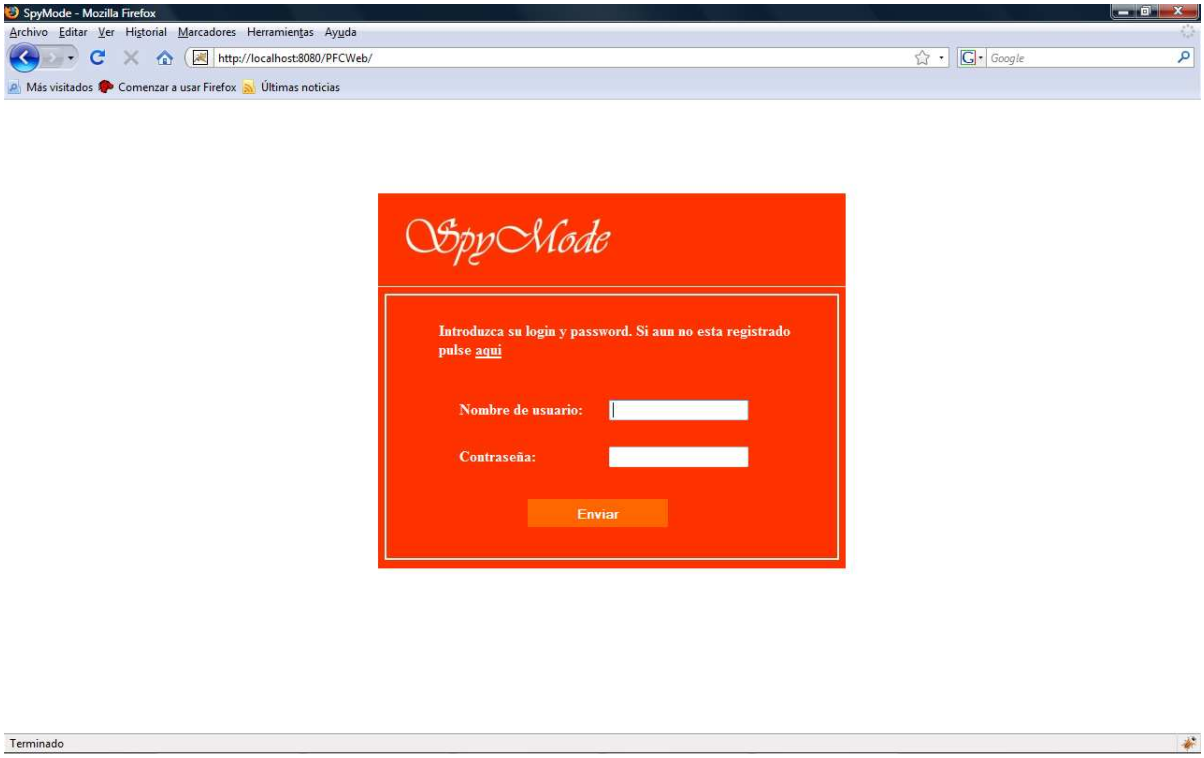


Figura 15: Página de autenticación de la aplicación Web

Al ser la primera vez que se accede al servicio el usuario deberá registrarse accediendo a la página de registro.

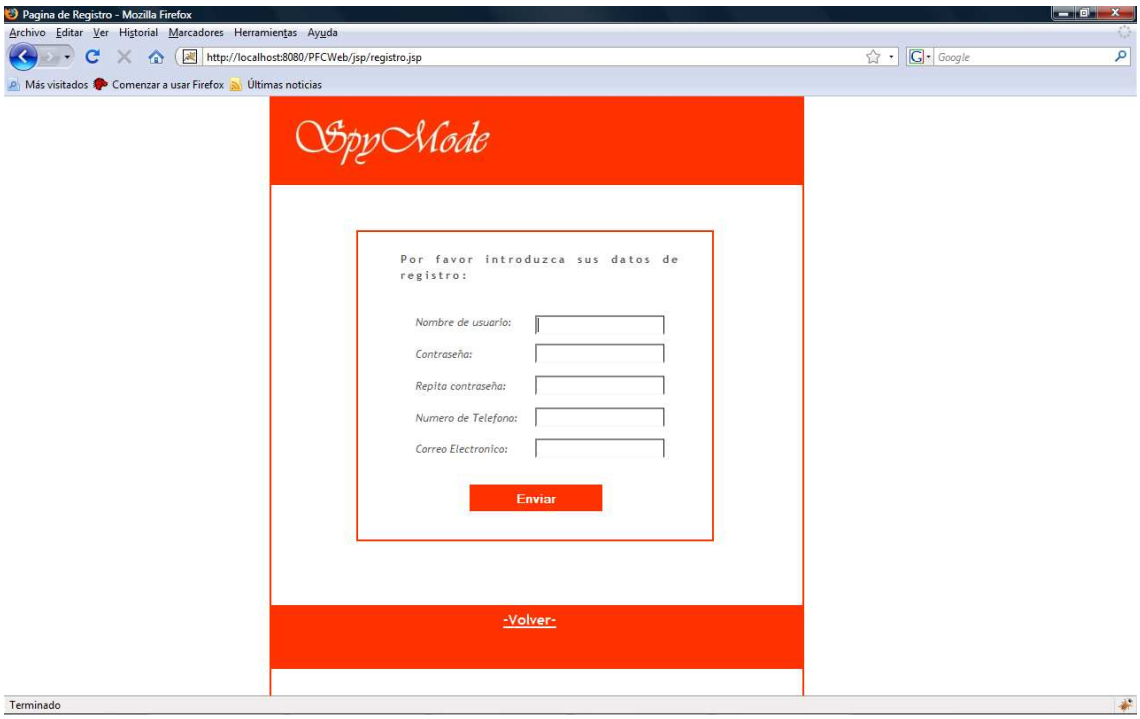


Figura 16: Página de registro de la aplicación Web

En esta página se deberá introducir los siguientes datos: nombre de usuario, contraseña, número de teléfono y dirección de correo electrónico. En caso de no introducir algún dato, que la confirmación de la contraseña no sea correcta o que la dirección de correo electrónico no sea correcta aparecerán los siguientes mensajes de error.

Figura 18: Detalle de mensajes de aviso en el proceso de registro en la aplicación Web

Figura 17: Detalle de mensajes de aviso en el proceso de registro en la aplicación Web

Figura 19: Detalle de mensajes de aviso en el proceso de registro en la aplicación Web

Una vez cumplimentado el formulario de registro se deberá pulsar el botón de envío para añadir al usuario a la base de datos. En caso de producirse algún error al insertar el usuario en la base de datos o que exista algún error en el servidor se mostrará la siguiente pantalla de error.

Figura 20: Pantalla de error en la aplicación Web

Pulsando el botón *"Volver"*, se mostrará nuevamente la página de inicio. Si el registro se ha realizado con éxito se mostrará la siguiente página.

Figura 21: Pantalla de éxito en la aplicación Web

Una vez realizado el registro el usuario podrá acceder a la aplicación móvil y hacer uso del servicio de detección de movimiento con envío de imágenes a través de Internet.

Al acceder a la aplicación móvil se muestra la siguiente pantalla de inicio.

Figura 22: Pantalla de inicio en la aplicación móvil

Si el usuario desea que la aplicación móvil se conecte a Internet deberá introducir el número de teléfono con el cual se ha registrado y pulsar el botón *"Si"*. Si no desea usar éste servicio deberá pulsar el botón *"No"* y la aplicación seguirá funcionando normalmente.

Al introducir el número de teléfono y pulsar el botón, se enviará una petición al servidor, el cual comprobará si ese número corresponde a algún usuario registrado en cuyo caso el servidor devolverá una respuesta 200 OK y la aplicación se iniciará. En caso de que no exista ningún usuario registrado con ese número de teléfono el servidor devolverá una respuesta 403 FORBIDDEN y se mostrará un mensaje de error. En caso de error en el servidor, se devolverá un error 500 INTERNAL SERVER ERROR, mostrándose un mensaje de error.

<table>
<tr>
<td> 

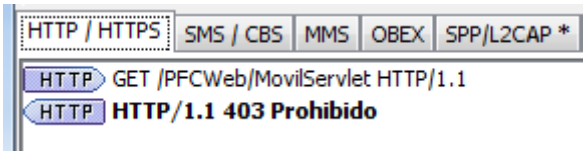
Figura 23: Pantalla de error en la aplicación móvil</td>
<td> 

Figura 24: Intercambio de información entre el teléfono móvil y el servidor Web</td>
</tr>
<tr>
<td>Pantalla de error al introducir un número de teléfono no registrado en la base de datos.</td>
<td>Intercambio de información entre el teléfono móvil y el servidor Web.</td>
</tr>
<tr>
<td> 

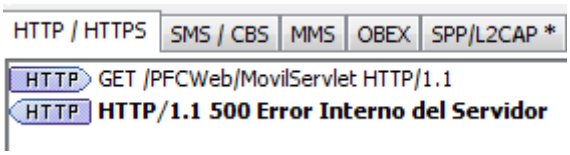
Figura 25: Pantalla de error al producirse un error del servidor</td>
<td> 

Figura 26: Intercambio de información entre el teléfono móvil y el servidor Web</td>
</tr>
<tr>
<td>Pantalla de error al producirse un error en el servidor.</td>
<td>Intercambio de información entre el teléfono móvil y el servidor Web.</td>
</tr>
</table>

Figura 27: Pantalla que se mostrará si se ha realizado la autenticación con éxito

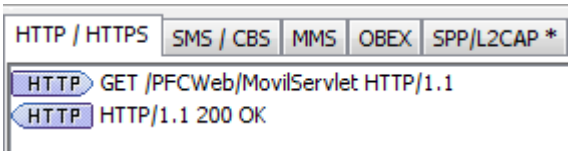


Figura 28: Intercambio de información entre el teléfono móvil y el servidor Web

Pantalla que se mostrará si se ha realizado la autenticación con éxito.	Intercambio de información entre el teléfono móvil y el servidor Web.

Cabe resaltar que si el número introducido no tiene 9 dígitos, se avisará al usuario con el siguiente mensaje de error.

Figura 29: Mensaje de aviso en caso de introducir un número de teléfono incorrecto

Una vez iniciada la detección de movimiento en el teléfono móvil, ésta, al detectar movimiento capturará una imagen y la enviará al servidor que la almacenará junto con datos relativos a la imagen tales como el número de teléfono móvil que la ha generado, la fecha y hora en la que se ha producido la detección y la ruta donde se guardará la imagen dentro del servidor.

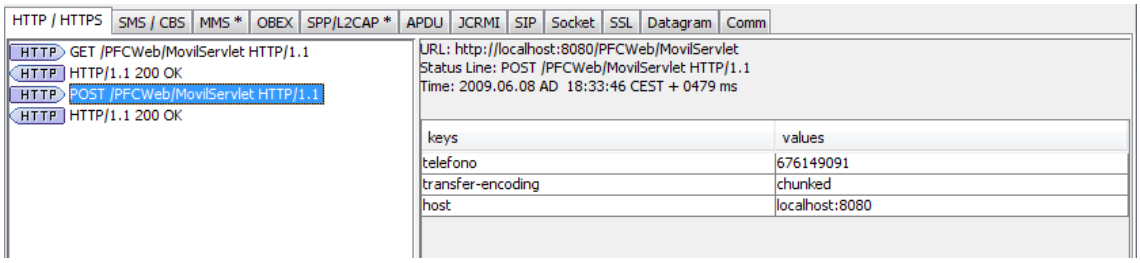


Figura 30: Detalle del envío de una imagen al servidor desde la aplicación instalada en el teléfono móvil

Detalle del envío de una imagen al servidor desde la aplicación instalada en el teléfono móvil.

El usuario podrá acceder a la aplicación Web introduciendo su nombre de usuario y contraseña, en caso de no introducir alguno de los dos campos del formulario se mostrarán los siguientes mensajes.

<table>
<tr>
<td> 
</td>
<td> 
</td>
</tr>
<tr>
<td>Figura 31: Detalle de mensaje de aviso en caso de no haber rellenado el campo “contraseña</td>
<td>Figura 32: Detalle de mensaje de aviso en caso de no haber rellenado el campo “nombre de “usuario”</td>
</tr>
<tr>
<td>Mensaje de error mostrado en caso de no haber rellenado el campo de “contraseña”.</td>
<td>Mensaje de error mostrado en caso de no haber rellenado el campo de “nombre de usuario”.</td>
</tr>
</table>

Si el usuario no se autentica de manera correcta se le mostrará este hecho con un mensaje de login incorrecto.

Figura 33: Detalle de pantalla en caso de realizar la autenticación incorrectamente

Si el usuario se autentica de manera correcta accederá a la siguiente página.

Figura 34: Página principal de la aplicación Web

Esta es la página principal del servicio. En la esquina superior derecha aparecerá el nombre de usuario y el número de teléfono móvil con el cual el usuario realizó el registro. En caso de que el usuario quiera abandonar el servicio deberá pulsar el botón *"Desconectar"* situado en la parte inferior de la página.

En esta página el usuario podrá acceder a las fotos que el teléfono móvil ha enviado en caso de haber detectado movimiento. En ella se mostrarán las imágenes junto con la fecha en la que se capturaron. Las imágenes se ordenarán en función de la fecha apareciendo en primer lugar las fotos capturadas más recientemente.

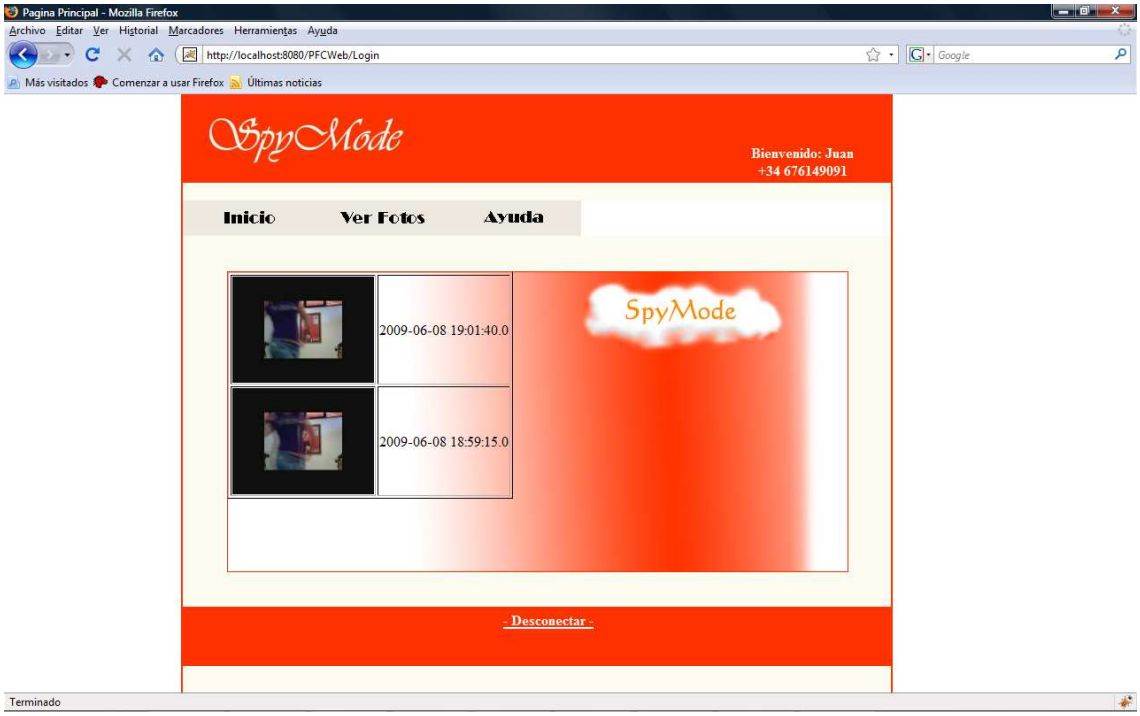


Figura 35: Visualización de imágenes enviadas por el teléfono móvil en la aplicación Web

También se podrá acceder a la página de ayuda donde se explica brevemente el funcionamiento del sistema.

Figura 36: Página de ayuda de la aplicación Web

Por último, el servidor, al recibir una imagen desde el teléfono móvil enviará la imagen capturada por correo electrónico a la dirección de correo electrónico especificada por el usuario al realizar el registro. El correo electrónico enviado será de la siguiente manera.

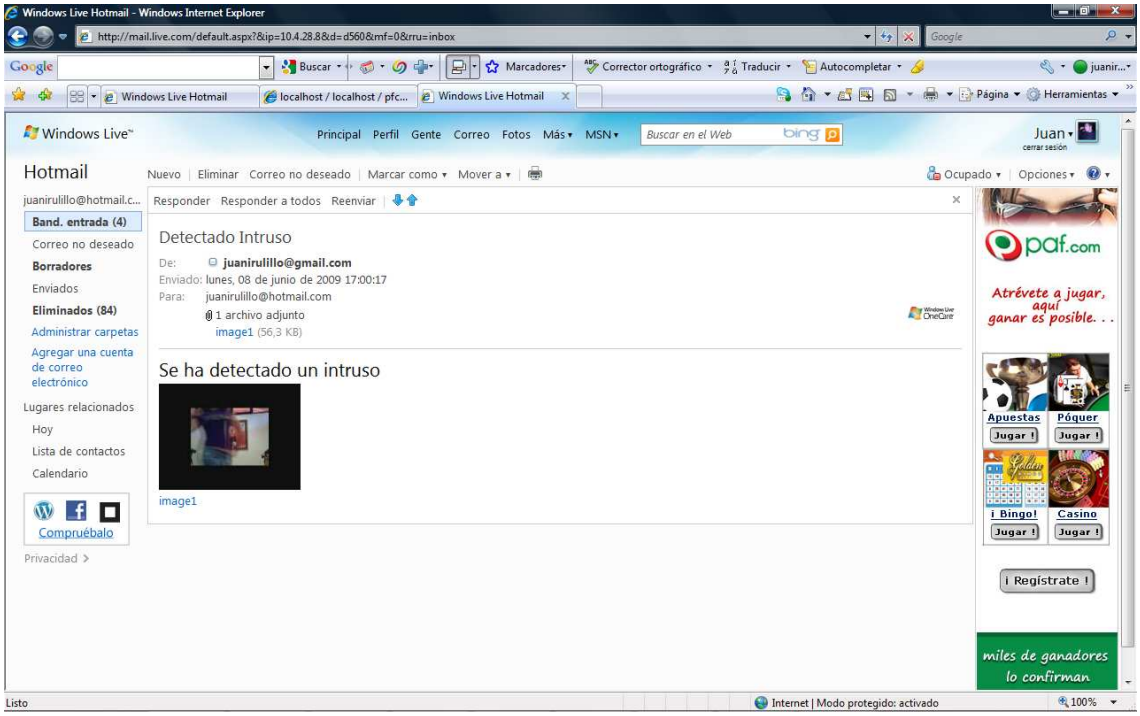


Figura 37: Correo electrónico enviado por el servidor Web en caso de haber recibido una imagen

7. Presupuesto y planificación.

7.1. Tareas.

Para realizar la planificación de este proyecto se ha decidido dividir dicho proyecto en las siguientes tareas:

Nombre de la tarea	Implementación del sistema móvil
Descripción	Implementación del sistema móvil (Especificado en el proyecto "*Desarrollo de un sistema de tele vigilancia a través de un dispositivo móvil*")
Duración	Especificado en el proyecto "*Desarrollo de un sistema de tele vigilancia a través de un dispositivo móvil*"
Recursos asignados	Ingeniero Técnico de Telecomunicaciones

Nombre de la tarea	Análisis y diseño del sistema servidor
Descripción	Análisis de requisitos, funcionalidad y tecnología a utilizar en el servidor Web.
Duración	4 días
Recursos asignados	Ingeniero Técnico de Telecomunicaciones

Nombre de la tarea	Formación en SQL, Servlets
Descripción	Adquisición de conocimientos previos para la implementación del servidor Web.
Duración	2 días
Recursos asignados	Ingeniero Técnico de Telecomunicaciones

Nombre de la tarea	Diseño e implementación de la base de datos
Descripción	Diseño de la base de datos del sistema en función de los requisitos especificados.
Duración	2 días
Recursos asignados	Ingeniero Técnico de Telecomunicaciones

Nombre de la tarea	Formación en HTML, CSS, AJAX, JavaScript
Descripción	Adquisición de conocimientos previos para la implementación del cliente Web.
Duración	6 días
Recursos asignados	Ingeniero Técnico de Telecomunicaciones

Nombre de la tarea	Implementación del servidor Web
Descripción	Implementación del servidor Web cumpliendo los requisitos de funcionalidad especificados.
Duración	6 días
Recursos asignados	Ingeniero Técnico de Telecomunicaciones

Nombre de la tarea	Implementación del cliente Web
Descripción	Implementación del cliente Web cumpliendo con los requisitos de funcionalidad y diseño especificados.
Duración	4 días
Recursos asignados	Ingeniero Técnico de Telecomunicaciones

Nombre de la tarea	Pruebas del sistema completo
Descripción	Pruebas del sistema servidor Web en conjunto con la aplicación móvil.
Duración	2 días
Recursos asignados	Ingeniero Técnico de Telecomunicaciones

Nombre de la tarea	Corrección de errores del sistema completo
Descripción	Corrección de los errores resultantes de las pruebas generales y nuevas pruebas para comprobar la correcta resolución de los mismos.
Duración	10 días
Recursos asignados	Ingeniero Técnico de Telecomunicaciones

Nombre de la tarea	Redacción de la memoria del proyecto
Descripción	Redacción de la memoria técnica del proyecto.
Duración	70 días
Recursos asignados	Ingeniero Técnico de Telecomunicaciones

7.2.Diagrama Gantt.

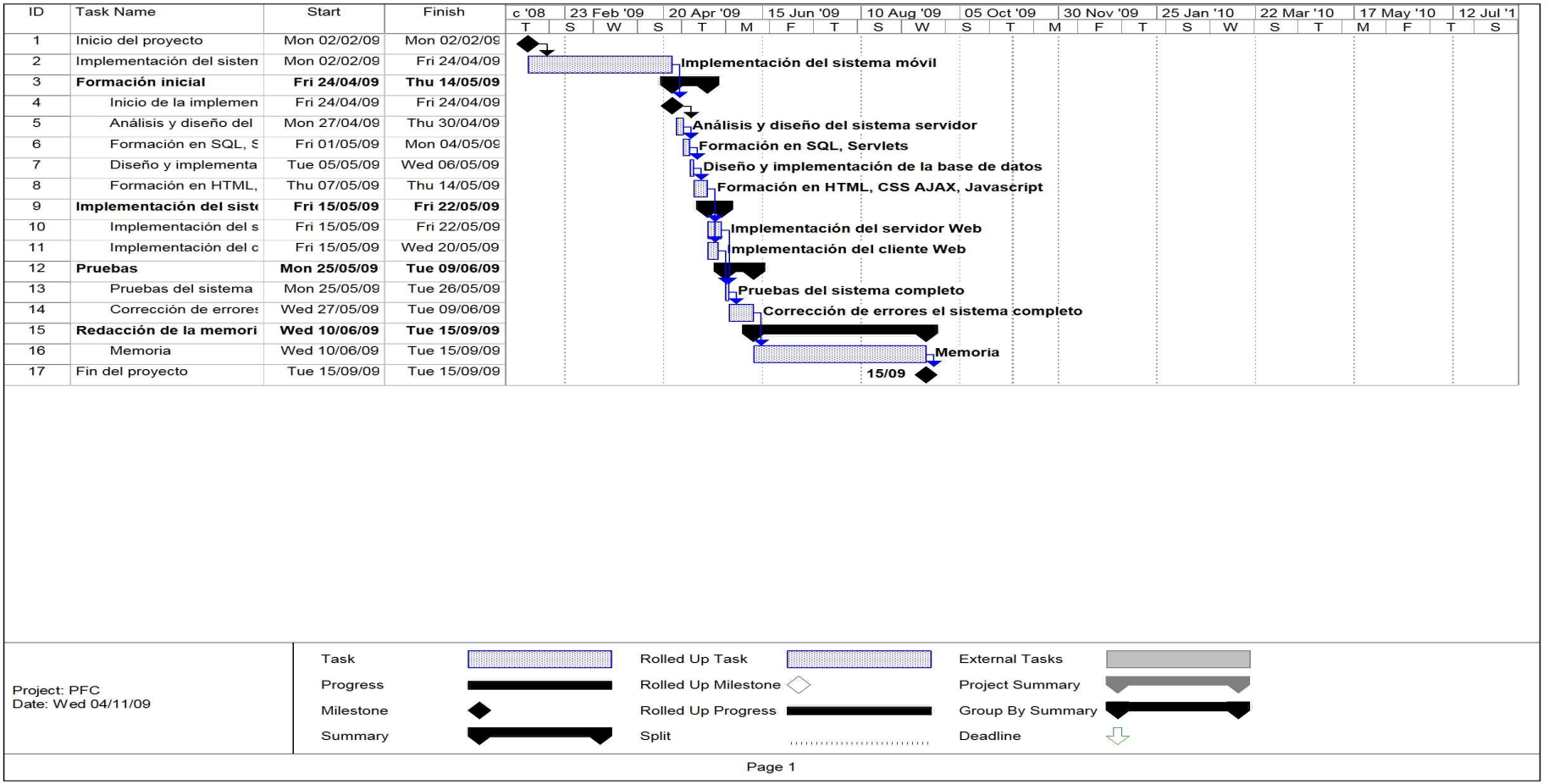


Figura 38: Diagrama GANTT del proyecto

7.3. Presupuesto.

Duración total del proyecto			
Días		106 días	
Horas		848 horas	
Recursos Utilizados			
Recurso	Ingeniero Técnico de Telecomunicaciones	Coste	30 euros/hora
Recurso	Ordenador personal Toshiba	Coste	685 euros
Recurso	Servidor Web Apache	Coste	0 euros
Recurso	Servidor Web Apache Tomcat	Coste	0 euros

Coste Total del proyecto		
Coste asociado a la mano de obra	Coste por horas	30 euros
	Número de horas de trabajo	848 horas
	Coste total	25.440 euros
Coste asociado a los materiales utilizados	685 euros	
Coste total del proyecto	26.120 euros	

Proyecto Global		
Nombre del proyecto	**Duración**	**Coste**
Desarrollo de un sistema de televigilancia a través de un dispositivo móvil	912 horas	28.619 euros
Integración de un sistema de televigilancia a través de un dispositivo móvil en un servidor Web	848 horas	26.120 euros
Duración total		1396 horas
Coste total		54.739 euros

8. Conclusiones y trabajos futuros.

8.1. Conclusiones.

Uno de los aspectos más delicados y sobre el que se ha debido de prestar una especial atención ha sido la comunicación entre el dispositivo móvil y el servidor Web. Inicialmente se barajaron dos tipos de tecnología de comunicación inalámbrica: Bluetooth y comunicación inalámbrica a través del protocolo HTTP, siendo esta ultima opción escogida.

A fecha de hoy, aproximadamente el 90% de los terminales móviles de nuestro país llevan implementado un sistema Bluetooth para poder comunicarse con otros dispositivos. Respecto al uso de Internet en un dispositivo móvil, aunque su empleo ha crecido en los últimos años llegando a alcanzar el 10% de las conexiones de banda ancha realizadas en España durante el último año, está limitado por norma general a terminales de alta gama y elevado coste. En este sentido las operadoras aun no ofrecen unas tarifas aceptables en relación calidad precio.

El análisis profundo de ambas tecnologías ha llevado a implementar la conexión HTTP en este sistema por las siguientes razones:

— El principal inconveniente del uso de la tecnología Bluetooth en este sistema radica en la implementación de una aplicación servidor instalada en un ordenador. Para atacar la posibilidad de la implementación de una aplicación Java que actuase de servidor Bluetooth se ha estudiado la librería Java, la cual implica la instalación del software BlueSoleil, la adquisición de un adaptador. Dichos elementos se han mostrado inseguros en las pruebas realizadas debido al Sistema Operativo sobre el cual actúan, en este caso Windows Vista, desaconsejando su uso ya que la mayoría de los ordenadores personales actuales llevan esta versión de Sistema Operativo.

— Una vez conocido el principal inconveniente del uso de la tecnología Bluetooth para la realización de una aplicación Java que actuase de servidor, explicado anteriormente, se analizaron otros factores de interés. Uno fue el alcance de este tipo de tecnología, que es generalmente de 10 metros, quedando en clara desventaja respecto a la conexión por banda ancha, con la cual se puede conectar desde cualquier punto. Esto obligaría a que la aplicación receptora se situara a una distancia determinada y reducida del sistema emisor limitando el uso del conjunto del sistema. Otro factor analizado fue el caudal proporcionado por cada una de las 2 tecnologías propuestas: Bluetooth asegura una velocidad de transferencia de 3 Megabits por segundo mientras que usando una conexión de banda ancha la velocidad que se obtiene es mucho menor y de mayor coste. No obstante cada vez más dispositivos móviles incluyen la conexión a redes WLAN, aprovechando la velocidad de transferencia que ofrece este estándar el cual puede alcanzar hasta los 54 Megabits por segundo con un rango de alcance mayor al Bluetooth. Nótese que para que el sistema funcione de manera eficiente se necesitan altas velocidades de transferencia.

— El hecho de usar conexiones HTTP para implementar la comunicación entre el dispositivo móvil y el servidor abre una puerta importante hacia el estudio e implementación de servidores Web y tecnologías Web derivadas tales como XHTML, CSS, JavaScript o AJAX. Este hecho también es favorable para el uso de bases de datos SQL y la visualización de estos datos en cualquier terminal con conexión a Internet.

Otro aspecto clave en la realización de este proyecto ha sido la elección de un servidor Web adecuado para el sistema. Se ha optado por un servidor Apache Tomcat ya que soporta el uso del lenguaje de programación Java a través de los Servlets con todas las ventajas que ello conlleva tales como la eficiencia en el acceso y el poder usar un gran número de API´s Java como por ejemplo el API JDBC para el acceso a bases de datos. Otro aspecto clave con gran peso en la elección final es que Apache Tomcat es un servidor Web gratuito.

La elección del tipo de base de datos es el último aspecto a destacar en la implementación del sistema. Se ha elegido una base de datos SQL por ser ésta un tipo de base de datos para la cual el lenguaje de programación Java proporciona la API JDBC que facilita la implementación de sistemas basados en este tipo de bases de datos.

Por todo lo expuesto en el anterior proyecto, para su diseño e implementación se ha necesitado profundizar y adquirir conocimientos sobre los siguientes aspectos:

- Protocolo HTTP, interacción cliente-servidor y estudio de las API´s Java para la conexión de terminales móviles mediante este protocolo.
- Uso de la tecnología Java para la implementación de servidores Web del tipo Apache Tomcat.
- Conocimientos en bases de datos SQL, lenguaje SQL, y su interacción con el servidor Apache Tomcat mediante la API JDBC.
- Conocimientos en tecnologías de diseño e implementación de sitios Web tales como XHTML, AJAX, CSS y JavaScript.
- Estudio de la API Java que da soporte al envío de correos electrónicos y sus protocolos relacionados (SMTP).

8.2. Trabajos futuros.

El principal trabajo futuro a realizar a partir del sistema implementado sería el desplegar dicho sistema en un entorno real ya que este proyecto está optimizado y probado únicamente en un entorno local. Esto conllevaría el diseño de scripts de mantenimiento de bases de datos, optimización de la carga servida por el servidor Web, pruebas de rendimiento y seguridad ante fallos de red. Para la parte del dispositivo móvil, conseguir el objetivo de desplegar el sistema en un entorno real tan sólo depende del terminal móvil utilizado, necesitando alguno de los terminales que han salido al mercado últimamente tales como el Nokia N97 así como una conexión a Internet con un ancho de banda más elevado a lo que actualmente ofrecen las operadoras.

También cabria destacar las siguientes mejoras para el sistema:

- Implementación de comunicación Bluetooth entre el dispositivo móvil y un ordenador y que desde allí la información se comparta con todos los terminales que se desee.
- Posibilidad de incluir streaming de video para poder visualizar los datos enviados desde el dispositivo móvil desde cualquier punto en tiempo real. Este hecho está limitado por la tecnología existente en los terminales móviles.

Apéndice A: Instalación de la aplicación en un entorno local.

A continuación se presenta una breve guía para poder ejecutar la aplicación en un entorno local. El sistema se ejecuta sobre el sistema operativo Windows Vista.

Paso 1: Instalación del servidor Web Apache-Tomcat

- Descargar la versión comprimida de Apache-Tomcat desde *http://tomcat.apache.org/*.
- Descomprimir Apache-Tomcat en el disco duro.
- Arrancar Apache-Tomcat desde la consola del sistema. Para ello hay que situarse en la carpeta *bin* y ejecutar el fichero *startup.bat*, para parar el servidor deberemos ejecutar el fichero *shutdown.bat*.

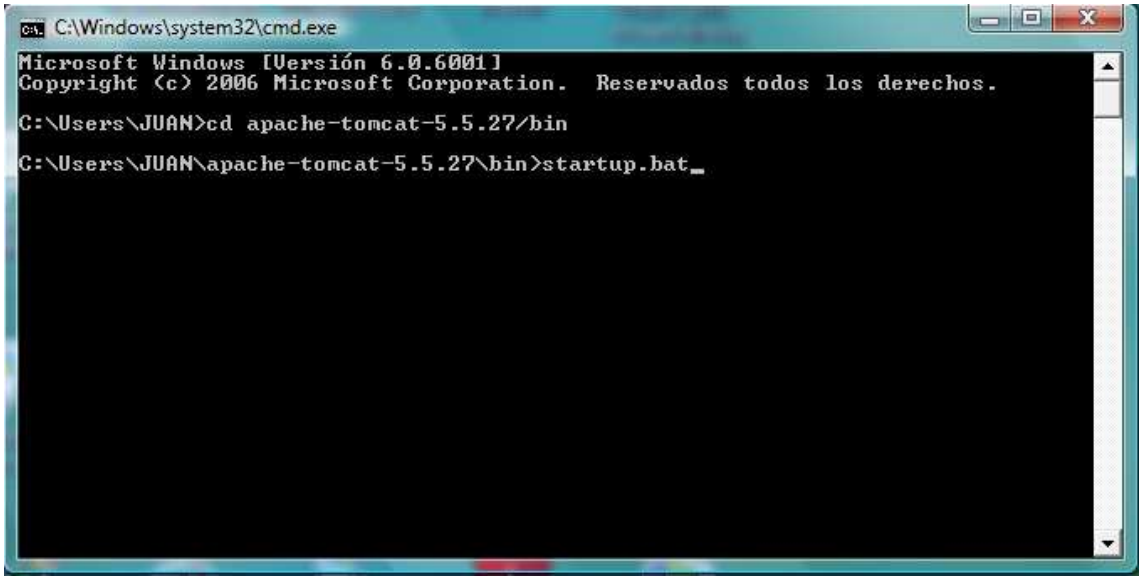


Figura 39: Detalle de consola para arrancar el servidor Web Apache-Tomcat

Una vez iniciado el servidor aparecerá la consola de Apache-Tomcat con los siguientes mensajes:

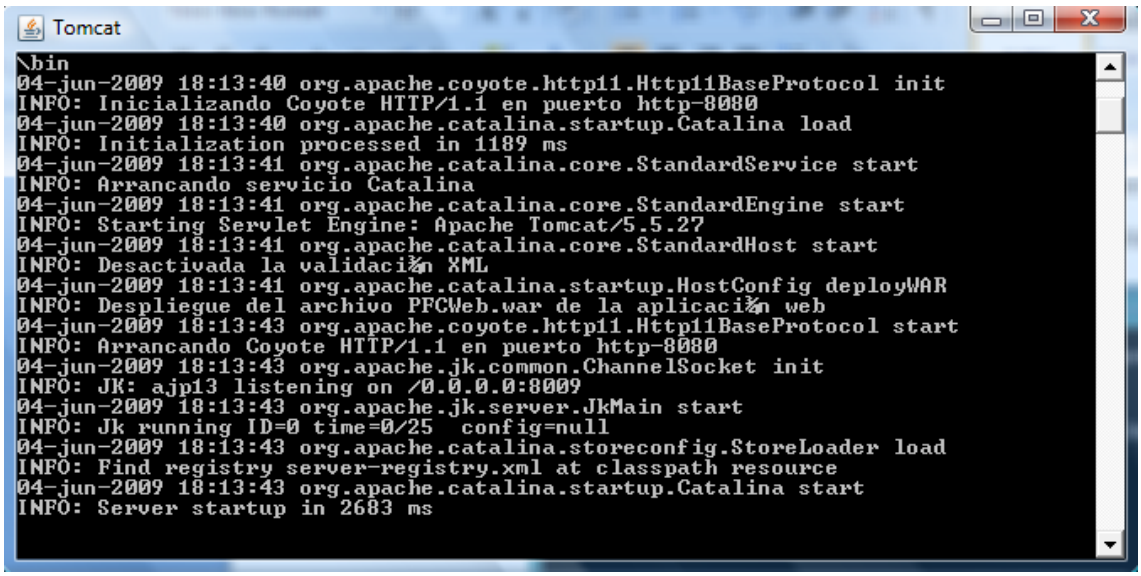


Figura 40: Detalle de la consola del servidor Web Apache Tomcat

En esta consola se podrán ver los posibles errores que se produzcan en el servidor así como trazas que ponga el desarrollador de la aplicación Web.

— Comprobar si se ha iniciado el servidor Web abriendo un navegador y accediendo a la URL http://localhost:8080. Si la instalación se ha realizado correctamente debería mostrarse la siguiente página:

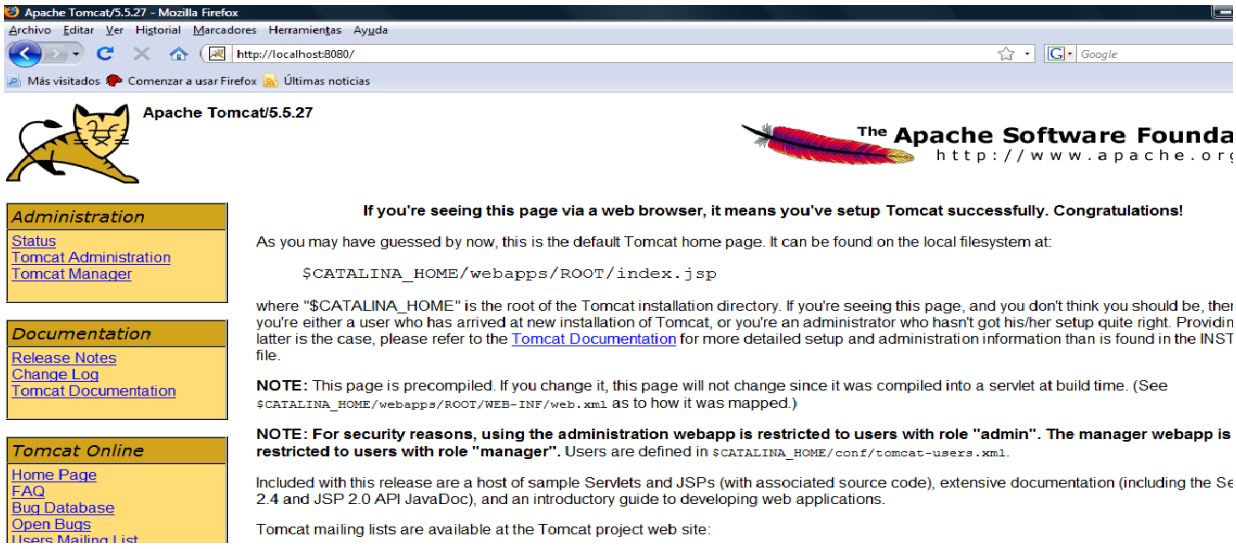


Figura 41: Página principal de Apache Tomcat

Paso 2: Instalación de la base de datos.

— Descargar el paquete XAMPP para Windows desde http://www.apachefriends.org/en/xampp.html, con este paquete instalaremos principalmente un servidor Apache, un servidor de base de datos SQL y la aplicación de gestión de bases de datos phpMyAdmin.

— Instalar XAMPP siguiendo las instrucciones del instalador.

— Una vez instalado XAMPP, ejecutar el programa con privilegios de administrador. Aparecerá la siguiente ventana:

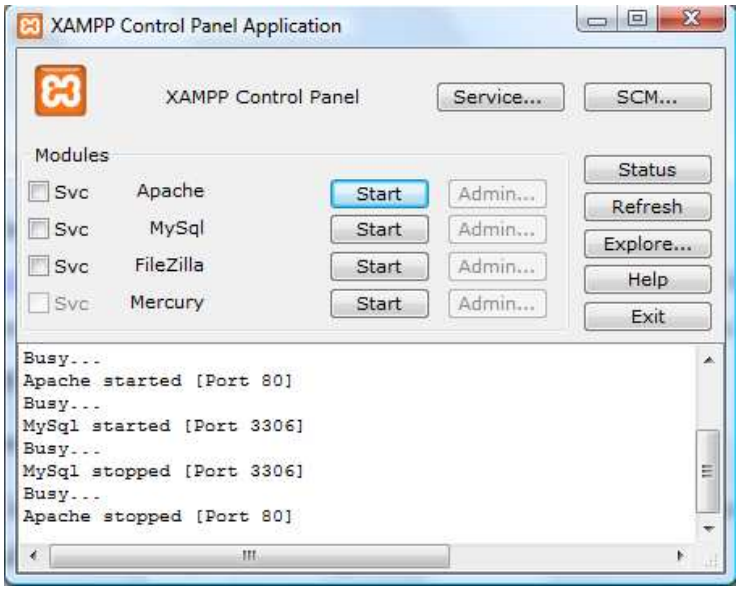


Figura 42: Consola de administración de XAMPP

Para iniciar Apache y MySQL se debe arrancar en primer lugar Apache y posteriormente MySQL.

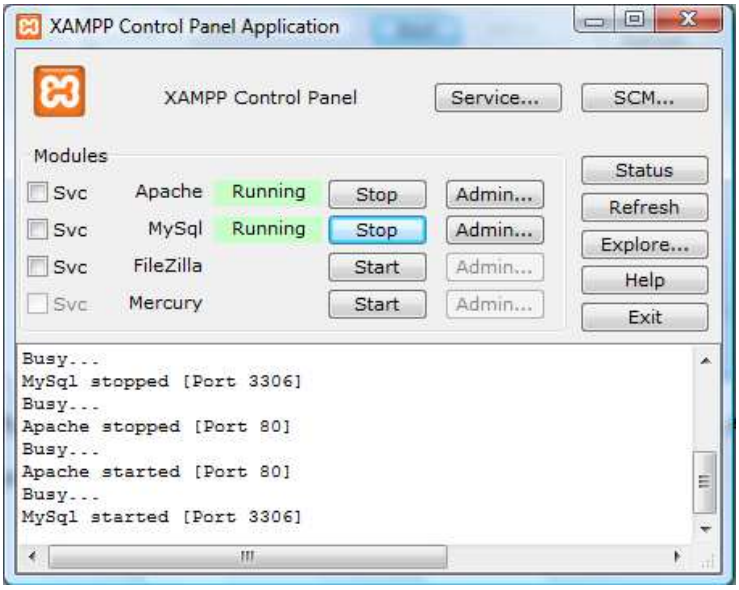


Figura 43: Consola de administración de XAMPP con los servidores Apache y MySQL iniciados

— Iniciar phpMyAdmin pulsando el botón Admin de MySQL o abriendo un navegador con la URL http://localhost/phpmyadmin/.

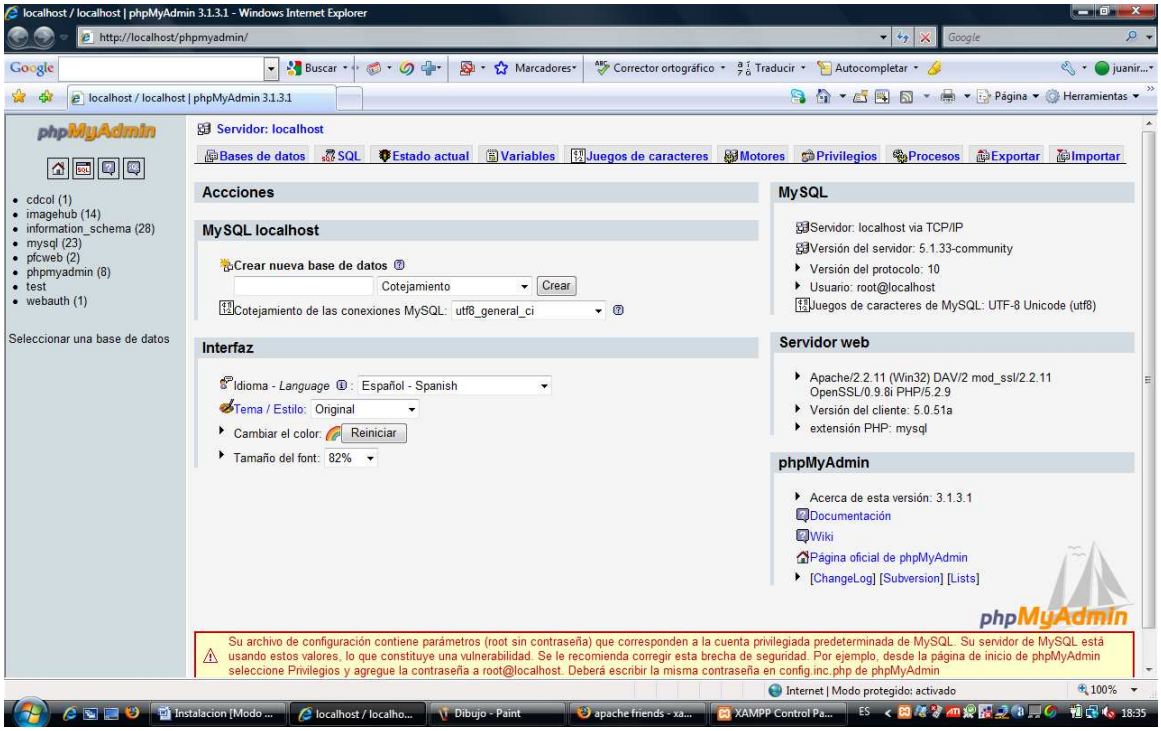


Figura 44: Página de inicio de phpMyAdmin

— Crear la base de datos con nombre *PFCWEB*. Una vez creada la base de datos, seleccionarla en la columna de la izquierda y pulsar la pestaña importar de phpMyAdmin.

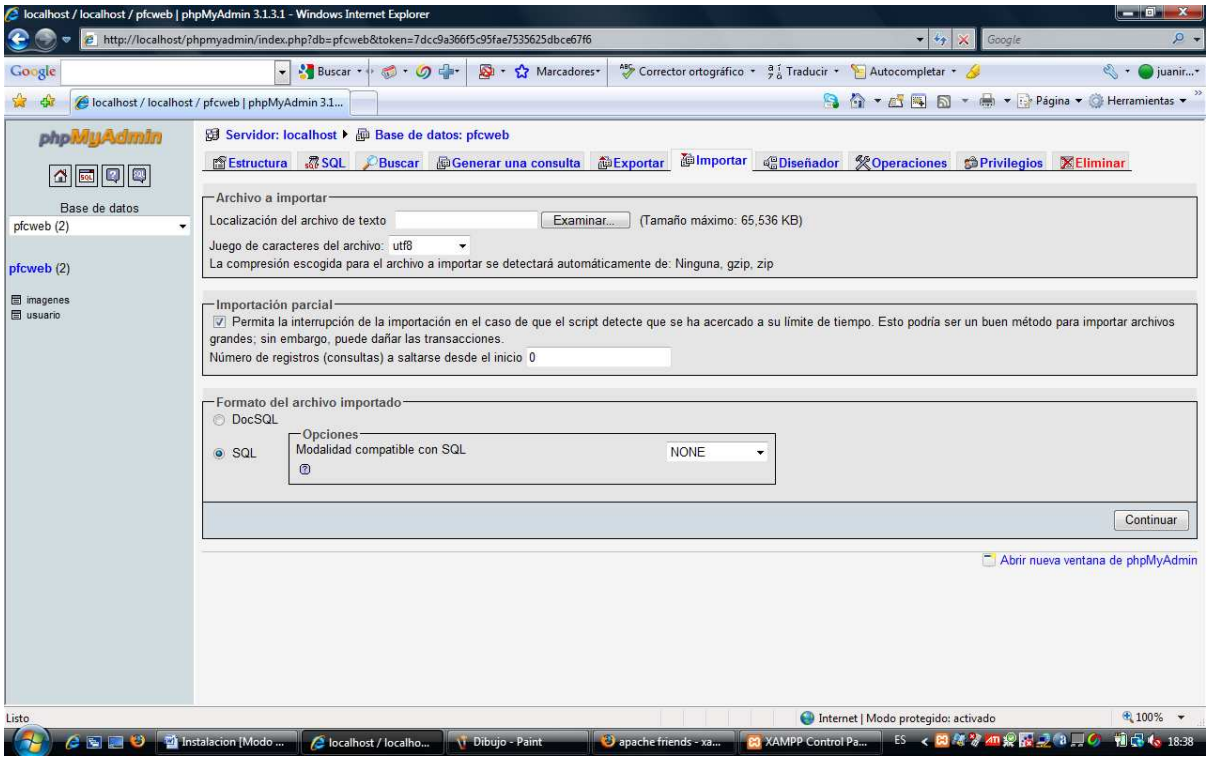


Figura 45: Página de importación de bases de datos en phpMyAdmin

— Importar el archivo usuario.sql incluido en el CD-ROM adjunto a este proyecto. Si la importación de la base de datos se ha realizado correctamente aparecerá lo siguiente:

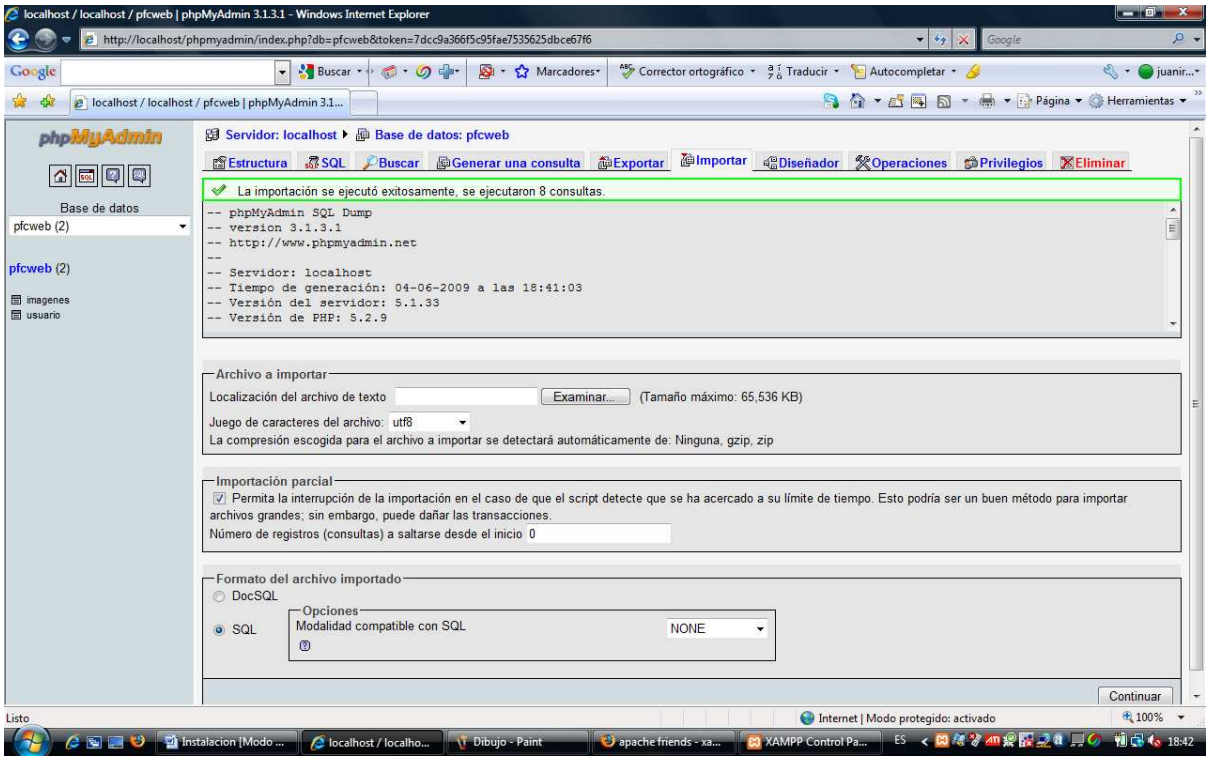


Figura 46: Página de importación de bases de datos correcta en phpMyAdmin

Paso 3: Desplegar la aplicación en el servidor Apache-Tomcat.

— Copiar el archivo *PFCWeb*.war contenido en el CD_ROM adjunto dentro de la carpeta *webapps* del sistema de directorios del servidor Web. (Importante: Para que la aplicación funcione correctamente se deberá configurar las variables *directory* y *path* de la clase *MovilServlet* para que apunten al servidor Web; por ejemplo *c: //rutainstalaciontomcat/webapps/PFCWeb/imgmvl*).

— Arrancar el servidor siguiendo los pasos detallados en el Paso 1 de éste apéndice.

— En un navegador abrir la URL http://localhost:8080/PFCWeb/ mostrándose si se han realizado los pasos exitosamente la siguiente página:

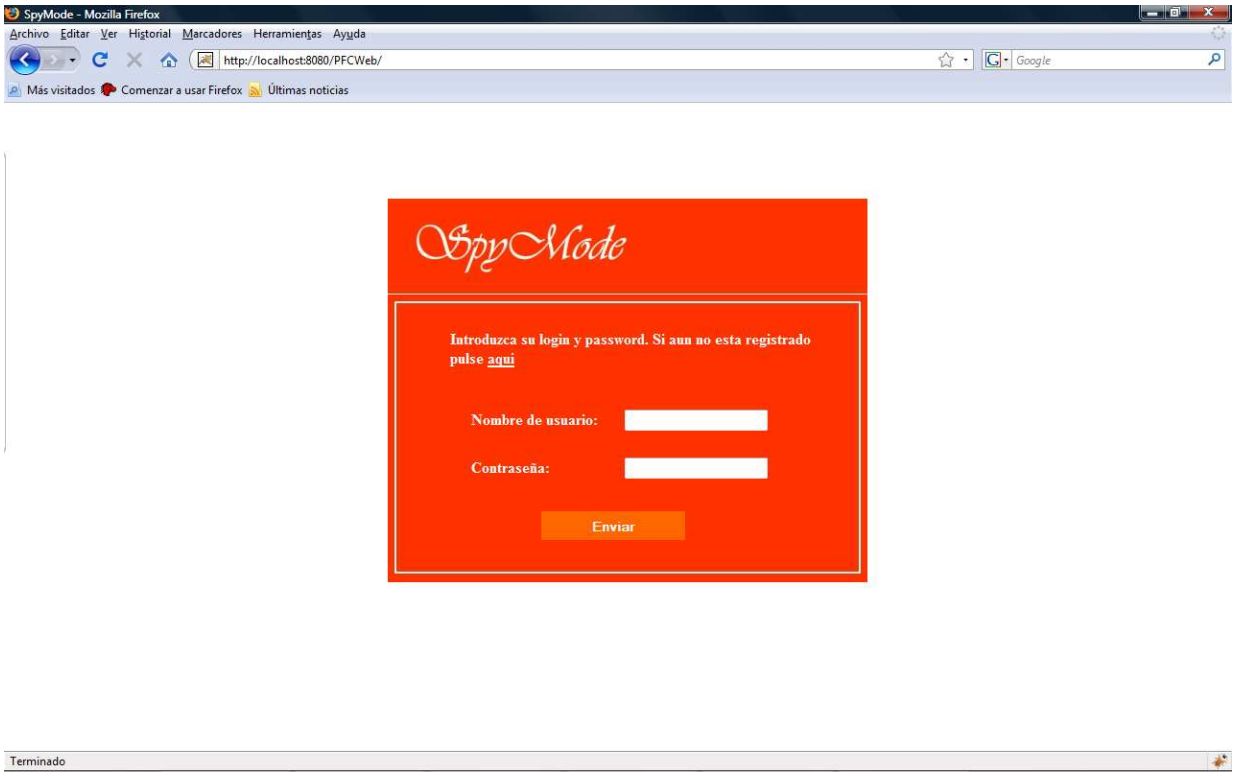


Figura 47: Página de autenticación de la aplicación Web

Bibliografía y Referencias

API´s:

- Servlets

  http://tomcat.apache.org/tomcat-5.5-doc/servletapi/index.html

- JavaMail

  http://java.sun.com/products/javamail/

- J2EE

  http://java.sun.com/j2ee/1.4/docs/api/index.html

- J2SE

  http://java.sun.com/j2ee/1.4/docs/api/index.html

- MIDP

  http://java.sun.com/javame/reference/apis/jsr118/

Libros:

- Javier Eguíluz Pérez, "*Introducción a AJAX*"
- Javier Eguíluz Pérez, "*Introducción a XHTML*"
- Javier Eguíluz Pérez, "*Introducción a CSS*"
- Jesús Sánchez Allende, Gabriel Huecas Fernández-Toribio, Baltasar Fernández Manjón, Pilar Moreno Díaz, "Java2 Iniciación y referencia", Osborne McGraw-Hill, 2001

Páginas Web:

- Página oficial de Apache

  http://httpd.apache.org/

- Página oficial de Apache-Tomcat

  http://tomcat.apache.org/

- Página oficial de MySQL

  http://www.mysql.com/

- Página oficial del proyecto XAMPP

  http://www.apachefriends.org/es/xampp.html

- Estándar CSS

  http://www.w3.org/Style/CSS/

Otras referencias:

- Wikipedia

  http://es.wikipedia.org/wiki/Wikipedia:Portada

- Desarrollo Web

  http://www.desarrolloweb.com/

- Adictos al Trabajo

  http://www.adictosaltrabajo.com/tutoriales/